掌握Robot自动化测试开源框架，提高重复功能测试工作效率

- Robot Framework自动化测试框架的基础用法
- 移动端、Web端、接口端等常见自动化测试Lib库的使用
- 自定义测试Lib库及测试案例常用编写技巧
- 自动化框架设计的思想以及其他类型自动化框架简介



Robot Framework自动化测试框架核心指南

张永清 著

清华大学出版社



Robot Framework自动化测试框架核心指南

张永清 著

清華大学出版社
北 京

内 容 简 介

Robot Framework 是目前众多自动化测试工具或者框架中一个非常流行的开源框架，致力于解决重复功能测试劳动所带来的高额成本，将自动化测试大众化、简单化、通俗化，让更多没有编程基础的人也能成功地完成自动化测试。

本书分 8 章，内容包括 Robot Framework 自动化测试框架的基础用法，框架的常用基础测试 Lib 库的使用（包含移动端、Web 端、接口端等常见自动化测试类型），自定义测试 Lib 库的编写，编写自动化测试案例常用的一些技巧，最后分享一下自动化框架设计的思想以及其他类型自动化框架简介。

本书适合 Robot Framework 初学者、软件测试工程师、软件测试经理阅读，也适合作为高等院校和培训学校相关专业课的配套参考书。

图书在版编目（CIP）数据

Robot Framework 自动化测试框架核心指南 / 张永清著.—北京：清华大学出版社，2019

ISBN 978-7-302-52392-5

Ⅰ. ①R… Ⅱ. ①张… Ⅲ. ①软件工具－测试－指南 Ⅳ. ①TP311.56-62

中国版本图书馆 CIP 数据核字（2019）第 038781 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：董　瑾

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：北京鑫丰华彩印有限公司
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**13　　**字　　数：**333 千字
版　　次：2019 年 4 月第 1 版　　**印　　次：**2019 年 4 月第 1 次印刷
定　　价：49.00 元

产品编号：081809-01

序

关于自动化测试的工具和框架其实有很多。自动化测试在测试 IT 行业中扮演着越来越重要的角色，不管是在传统的 IT 行业还是高速发展的互联网行业或是如今的大数据和大热的人工智能领域，都离不开测试，也更加离不开自动化测试。自动化测试已经发展了很多年，在很多大的互联网公司里他们不但有自己的自动化测试框架，甚至基本都建立了自己的自动化测试平台或者已经对外开放使用的自动化测试云平台，也就是说自动化测试几乎已经成为一名测试工程师必须掌握的一个技能，并且随着很多自动化测试工具和框架的不断发展和完善，自动化测试也变得越来越简单。Robot Framework 是目前众多自动化测试工具或自动化测试框架中一个非常流行的开源框架，致力于解决重复功能测试劳动所带来的高额成本，将自动化测试大众化、简单化、通俗化，让更多没有编程基础的人也能成功地完成自动化测试，降低自动化测试带来的学习成本。

2016 年年初，作者在规划写这本书的时候，其实已经在开始谋划做职位转型，从一名软件测试工程师转型为一名软件开发工程师，在此之前作者曾经在测试职位上打拼了 8 年多，经历了手工功能测试、自动化测试、性能测试。在写这篇序时，作者已经在转型后的软件开发工程师的岗位上工作了 1 年多，之所以说作者自身的经历，其实主要是想告诉每一位想从事自动化测试的读者，只要想去做或者想去转型，时间和年纪都不会是太大问题，哪怕你已经 30 岁或者 30 多岁了，都可以重新开启一个新的奋斗起点。作者写这本关于自动化测试框架的书，除了分享一些自身浅薄的经验外，还有一个目的就是想鼓励更多还在从事手工功能测试的读者去学习自动化测试，去从事自动化测试。

这本书并不是完全面向初学者来进行设计的，更多的是比较适合有一定自动化基础的朋

友。在本书的后半部分，更多的谈到如何去设计一个自动化测试框架， 对于一个刚刚接触自动化测试的朋友来说，刚开始可能会稍显吃力，但是随着您对自动化测试的逐步深入，相信您会越来越轻松、越来越喜欢。

感谢孟瑞迪、Monica 等众多挚友在我最困难的时候给予我很多的帮助，正是有了大家的帮助才有了这本关于自动化测试框架的书。由于作者水平有限，书中难免会存在一些不足之处，恳请读者提出宝贵的意见和建议。

作者于南京

2019 年 1 月

目　录

第 1 章 初识Robot Framework

Robot Framework 是一款基于 Python 编程语言设计的、可扩展的、关键字驱动模式的测试自动化框架，具备良好的可扩展性，可以通过 XML-RPC 服务扩展支持其他的常用编程语言，可以同时测试多种类型的客户端或者接口，可以支持进行分布式测试执行。

Robot Framework 具体的特点如下：

- 易于使用，采用表格式输入语法以及统一的测试用例（Test Case，也叫测试案例）格式。
- 重用性好，可以利用现有关键字来组合新的用户自定义关键字。
- 支持资源文件，支持多种变量类型，包括字符串变量、List 列表变量、Dictionary 字典变量等。
- 测试用例执行结果报告和日志采用 HTML 格式，易于阅读和邮件转发。
- 提供标签以分类来选择将被执行的测试用例，使得测试用例的选择更加灵活。
- 支持 Web 界面测试、Web 接口服务测试、GUI 测试、多种终端测试。
- 支持多种数据库的操作，包括常用的关系型数据库、非关系型数据库。
- 易于扩展自定义的 Lib 库，可以通过 Python 或者 Java 等其他开发语言来动态扩展 Lib 库。

Robot Framework 自动化测试框架的组成如图 1-0-1 所示。

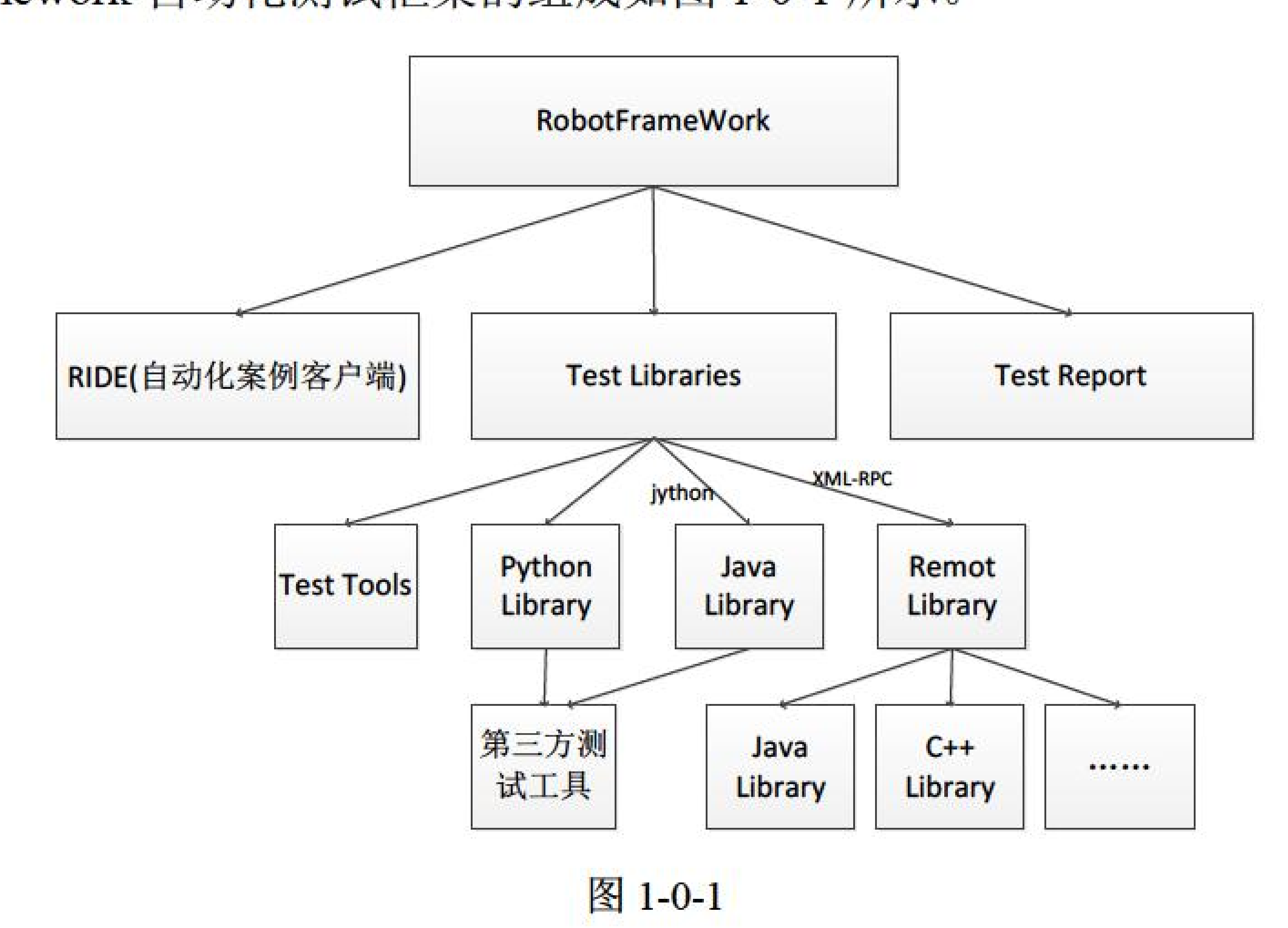


图 1-0-1

- 集成了很多流行的自动化测试工具，比如 Appium、Selenium 等。

- 通过 Jython 的方式，使得以 Python 为主的 Robot Framework 自动化测试框架无缝地与 Java 语言进行完美集成，也可以通过 XML-RPC 远程调用协议来支持 Java 或者 C++ 等流行的编程语言，使对 Python 语言不熟悉的编程爱好者也可以编写自定义的 Library 库。
- 在 Robot Framework 中，使用 Python 语言实现了自动化测试用例编写的客户端 RIDE，使用 RIDE 可以非常简单地完成自动化测试用例的编写，也可以使用 RIDE 完成用户层面的关键字 API 的封装，使得不懂任何编程语言的自动化测试爱好者也可以封装自己的 API 关键字。

Robot Framework 除了提供了我们上面提到的 Ride 外，还提供了很多常用的插件工具，如表 1-0-1 所示。

表 1-0-1 Robot Framework 常用的插件工具

工具插件	说明
Eclipse plugin	Robot Framework 为 Eclipse IDE 开发工具提供的插件，使得用户也可以在 Eclipse 上编写自动化测试用例，GitHub 地址为： https://github.com/NitorCreations/RobotFramework-EclipseIDE/wiki
Robot Plugin for IntelliJ IDEA	和 Eclipse plugin 类似，是为另一个常用的 IDE 开发工具 IntelliJ IDEA 提供的插件，使得用户也可以在 IntelliJ IDEA 上编写自动化测试用例，GitHub 地址为：https://plugins.jetbrains.com/plugin/7430-robot-plugin
Jenkins plugin	这是一个 Jenkins 上使用的插件，这个插件可以使得 Robot Framework 完美地集成在当今非常流行的持续集成工具 Jenkins 上，插件的访问网址为：https://wiki.jenkins-ci.org/display/JENKINS/Robot+Framework+Plugin
Maven plugin	这是 Robot Framework 提供的 maven 仓库插件，可以通过访问网址"http://robotframework.org/MavenPlugin/"获取，当前最新的版本为 1.4.7
Ant task	这是为另一个打包工具 ant 提供的执行插件，使得 Robot Framework 可以通过 ant 的方式来运行。可以通过访问网址"http://code.google.com/p/robotframework-ant"/获取该插件
Pabot	Robot Framework 提供的并发执行器，也就是我们通常说的多线程并发执行模式，可以通过在 Windows 的 cmd 下执行 pip install -U robotframework-pabot 命令进行在线安装，也可以通过访问 GitHub 网址"https://github.com/mkorpela/pabot"进行下载，然后离线进行安装
Atom plugin	Robot Framework 为支持 Atom 而开发的插件，可以通过访问网址"https://atom.io/packages/language-robot-framework"进行下载

1.1 如何创建一个自动化测试项目

一个 Robot Framework 项目其实就和一个我们平时熟知的单元测试项目结构基本是一样的，也包含了测试套件和测试用例的概念。我们可以对 Robot Framework 项目结构做如图 1-1-1 所示的划分。



图 1-1-1

1.1.1　创建测试项目

在 Robot Framework 中，Ride 是一款用 Python 语言实现的用来做自动化测试用例编写的客户端工具。通过访问网址“https://pypi.org/project/robotframework-ride/”即可下载 Ride 工具包进行离线安装，也可以通过在 Windows 的 cmd 命令行中输入“pip install robotframework-ride”进行在线自动安装。安装完成后打开 Ride，选择菜单栏 File→New Project，在 Name 文本框中输入项目名称，此处 Type 我们选择 Directory，单击 OK 按钮，即可创建成功，如图 1-1-2 所示。

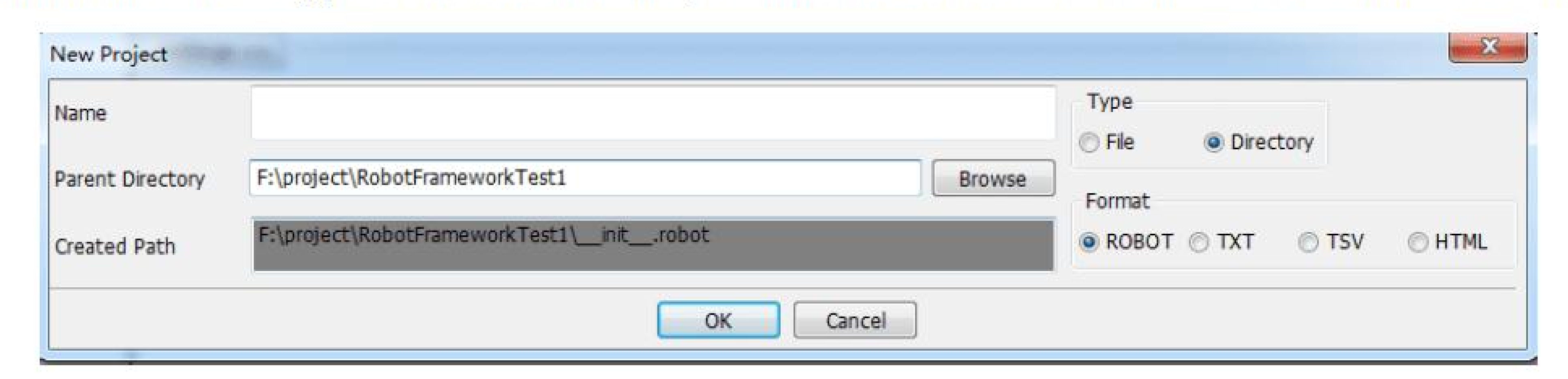


图 1-1-2

存储格式支持多种类型，如表 1-1-1 所示。

表 1-1-1　存储格式支持的类型

储存类型	说明
Type	项目存储方式：文件形式或者目录形式，一般建议选择目录形式
Format	文件存储格式：提供了 ROBOT（默认格式）、TXT、TSV 和 HTML 四种格式

1.1.2　创建测试套件

选择上面我们创建好的项目，右击鼠标键，选择 New Suite 选项，输入测试套件名称，即可创建成功，如图 1-1-3 所示。

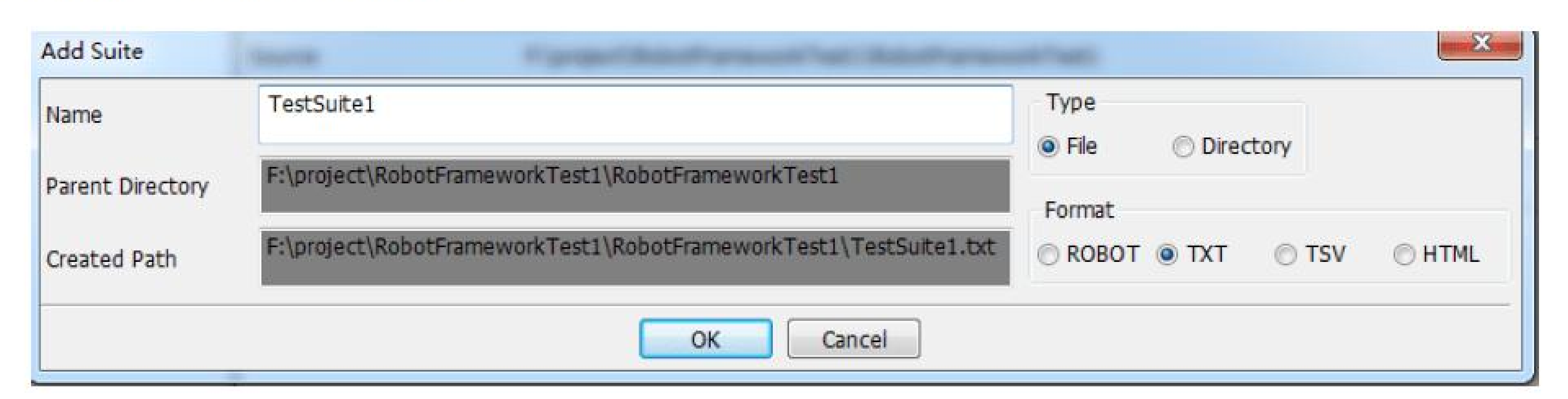


图 1-1-3

1.1.3 创建测试用例

选择上面我们创建好的测试套件，右击，选择 New Test Case 选项，输入用例名称，单击 OK 按钮，即可创建成功，如图 1-1-4 所示。

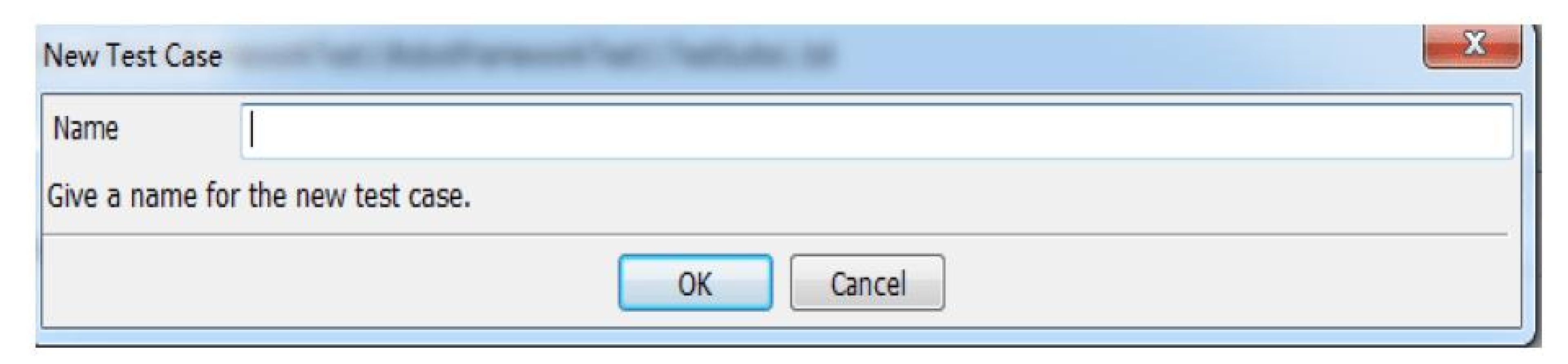


图 1-1-4

创建成功后，即可看到下面的用例编写表格，如图 1-1-5 所示。通过此表格，我们就可以编写测试用例了。

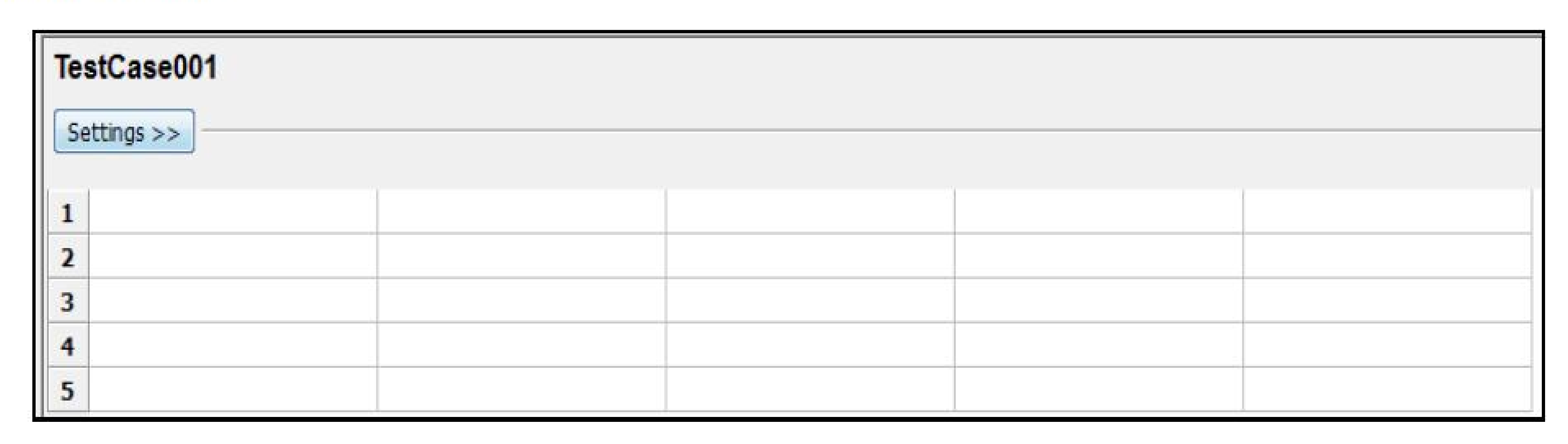


图 1-1-5

1.2 Robot Framework 基础关键字

1.2.1 如何搜索 Robot Framework 的关键字

有两种方式可以快速地打开 RIDE 的关键字搜索对话框。

（1）选择菜单栏中的 Tools→Search Keywords 选项，然后会出现如图 1-2-1 所示的关键字搜索对话框，这个对话框就类似于提供了一个关键字的 API 功能（提供了关键字的名称、关键字的来源库、关键字的使用描述和关键字的参数）。

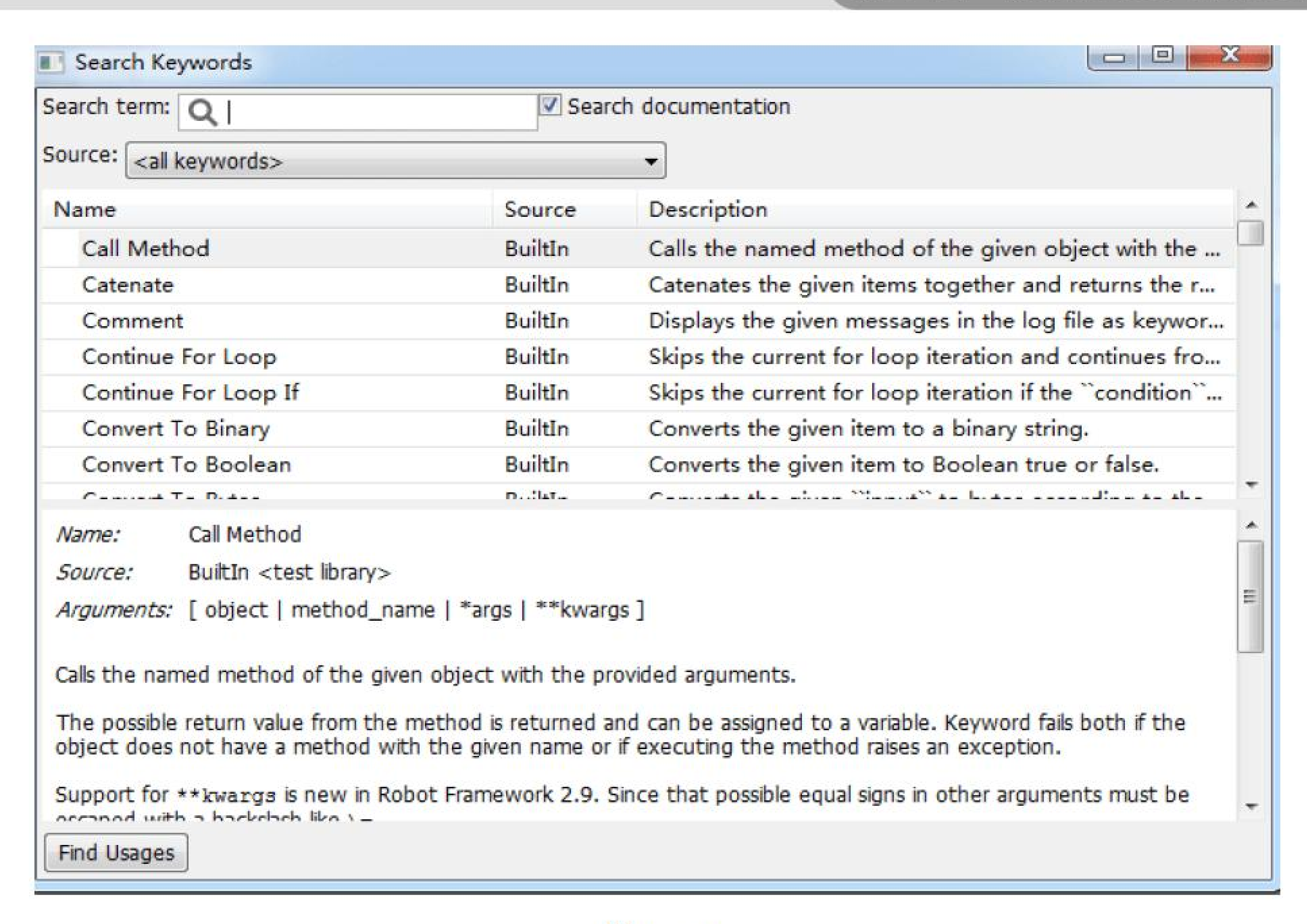


图 1-2-1

（2）直接按 F5 快捷键，就可以自动弹出我们需要的关键字搜索框。

1.2.2　关键字 log

Log 关键字其实就等同于 Python 语言中的 print 函数，可以输出我们想要输出的内容（也就是我们在编程语言中常说的日志输出），比如我们在 test case 中输入如图 1-2-2 所示的内容。

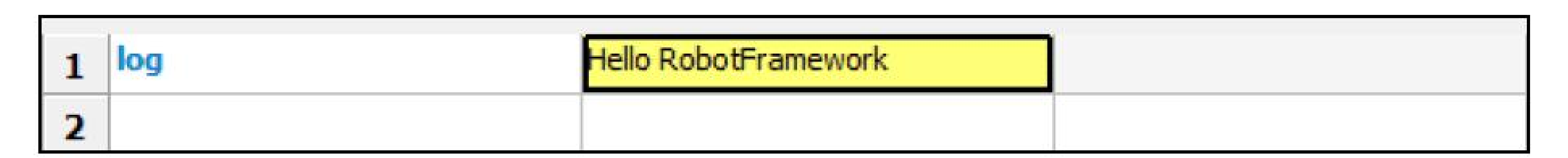


图 1-2-2

勾选我们的测试用例，单击菜单栏 Tools→Run Tests（或者直接快捷键 F8）来执行这条测试用例，如图 1-2-3 所示。

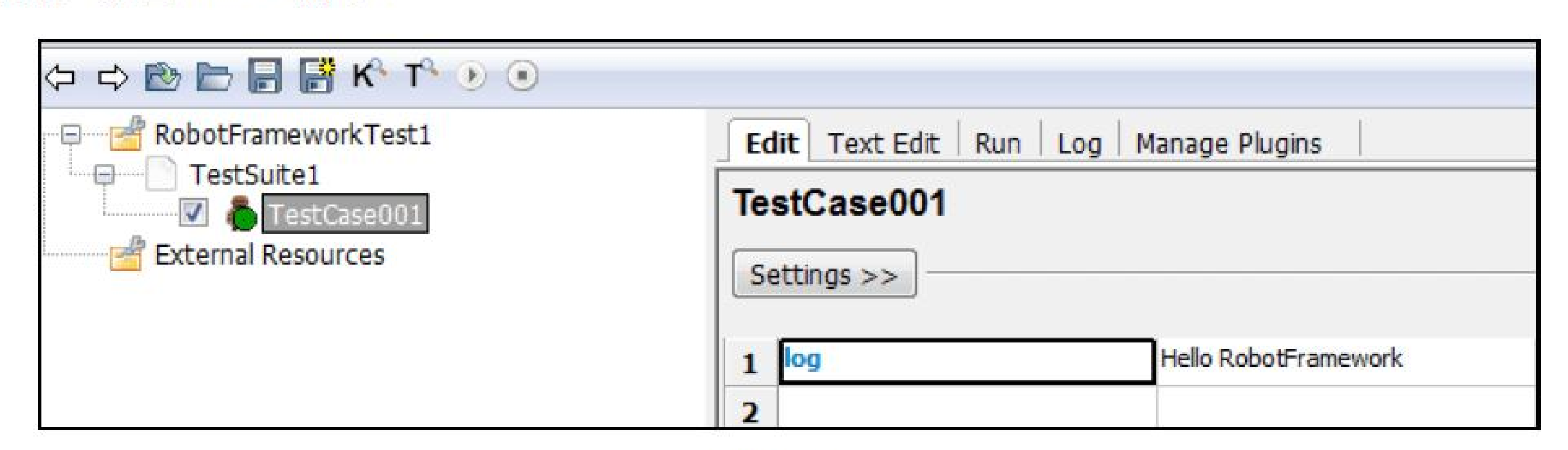
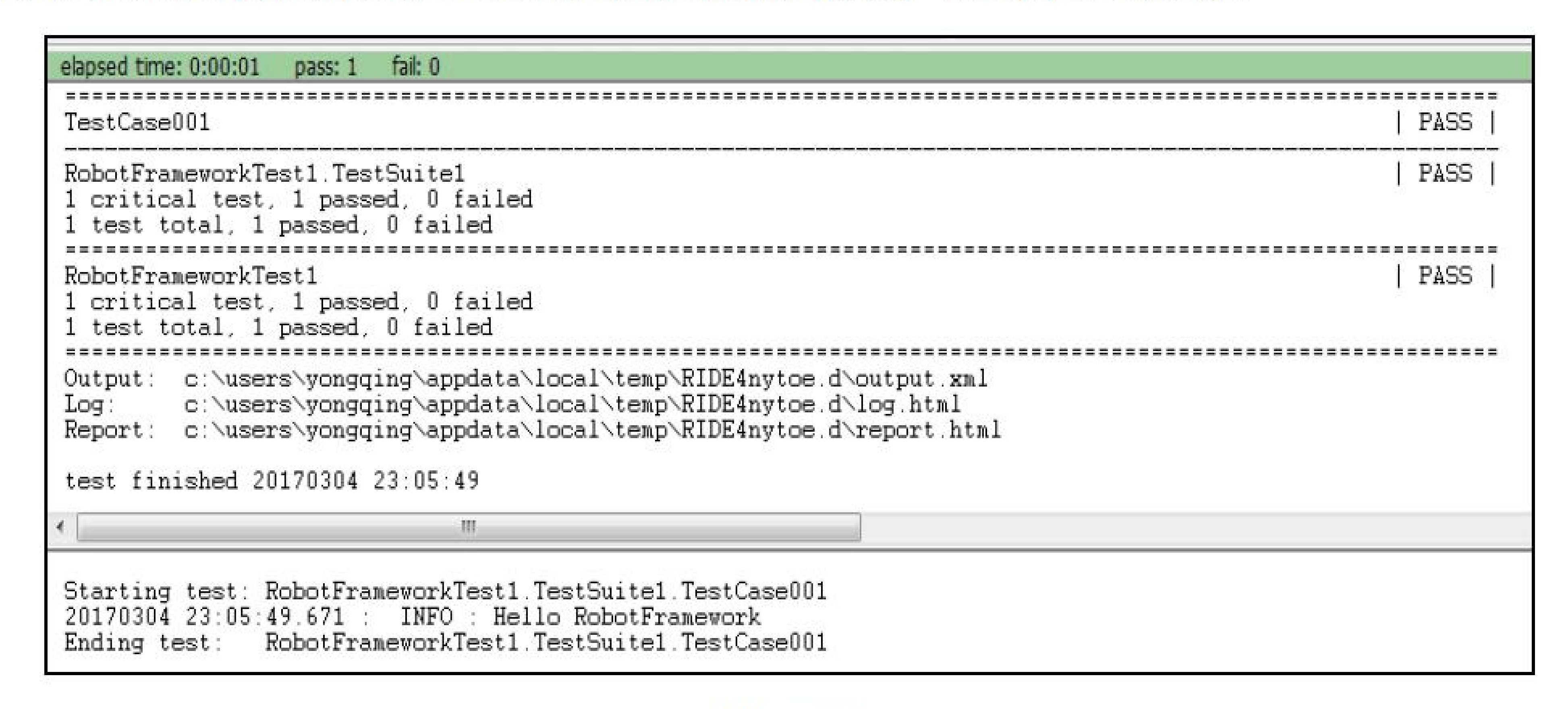


图 1-2-3

执行完成后，切换到 Run 标签，可以看到用例执行的结果。通过运行结果可以看到输出

了我们想要输出的信息 INFO : Hello RobotFramework，如图 1-2-4 所示。



```
elapsed time: 0:00:01    pass: 1    fail: 0
==============================================================================
TestCase001                                                           | PASS |
------------------------------------------------------------------------------
RobotFrameworkTest1.TestSuite1                                        | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
RobotFrameworkTest1                                                   | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  c:\users\yongqing\appdata\local\temp\RIDE4nytoe.d\output.xml
Log:     c:\users\yongqing\appdata\local\temp\RIDE4nytoe.d\log.html
Report:  c:\users\yongqing\appdata\local\temp\RIDE4nytoe.d\report.html

test finished 20170304 23:05:49

Starting test: RobotFrameworkTest1.TestSuite1.TestCase001
20170304 23:05:49.671 :  INFO : Hello RobotFramework
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase001
```

图 1-2-4

1.2.3 如何在用例中定义一个变量

我们可以通过 Set Variable 来定义一个变量，比如我们定义一个变量 var1，并且将这个变量赋值为 Robot，然后将这个变量用 log 输出，如图 1-2-5 所示。

${var1}	Set Variable	Robot
log	${var1}	

图 1-2-5

执行结果如图 1-2-6 所示。

```
==============================================================================
TestCase001                                                           | PASS
------------------------------------------------------------------------------
RobotFrameworkTest1.TestSuite1                                        | PASS
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
RobotFrameworkTest1                                                   | PASS
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  c:\users\yongqing\appdata\local\temp\RIDE4nytoe.d\output.xml
Log:     c:\users\yongqing\appdata\local\temp\RIDE4nytoe.d\log.html
Report:  c:\users\yongqing\appdata\local\temp\RIDE4nytoe.d\report.html

test finished 20170304 23:16:32

Starting test: RobotFrameworkTest1.TestSuite1.TestCase001
20170304 23:16:32.589 :  INFO : Hello RobotFramework
20170304 23:16:32.589 :  INFO : ${var1} = Robot
20170304 23:16:32.599 :  INFO : Robot
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase001
```

图 1-2-6

1.2.4 如何快速查询某一个关键字的 API 说明

选中关键字，同时按住 Ctrl+Alt 组合键，即可显示该关键字的帮助 API 以及使用示例，如图 1-2-7 所示。

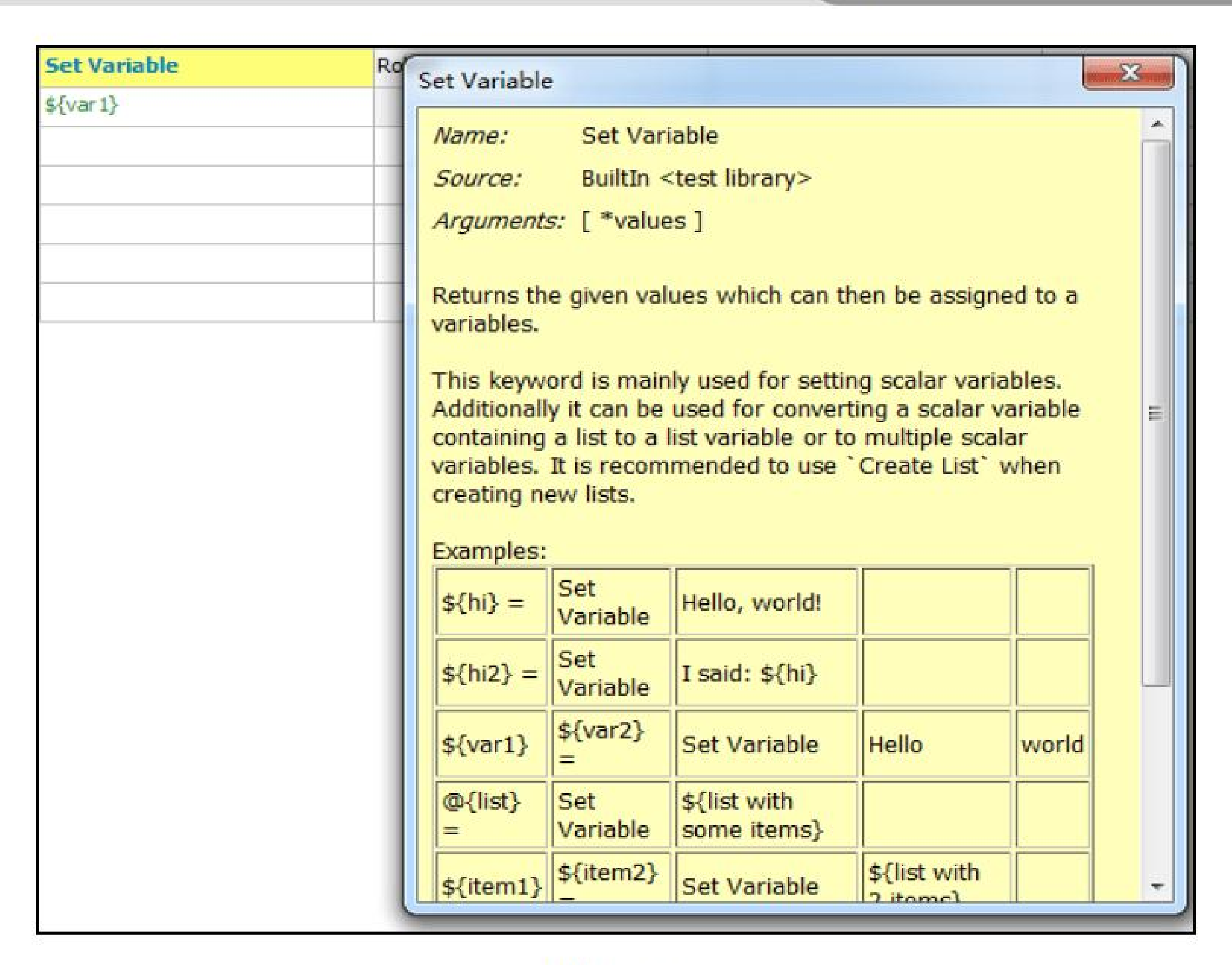


图 1-2-7

1.2.5　如何快速补全关键字

通过键盘输入关键字的前缀，然后同时按住 Ctrl+Alt+空格键，即可快速补全某个关键字，如图 1-2-8 所示。

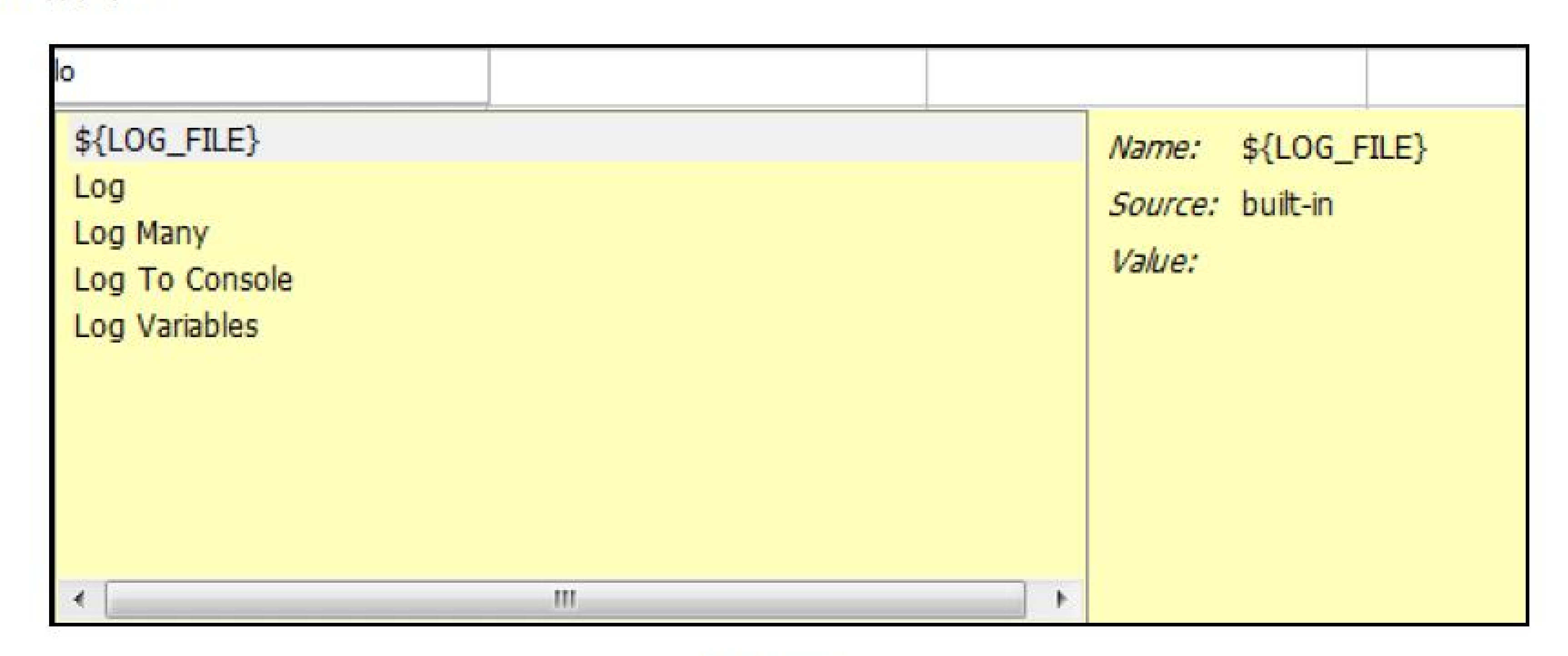


图 1-2-8

1.2.6　如何定义一个列表

此处我们说的列表，其实就等同于 Python 语言中的列表，是 Python 语言中常用的一种数据结构，也类似于 Java 语言中的 List。

在 Robot Framework 中，我们可以使用 Create List 来创建一个列表，比如我们定义一个列表 list1，并且在创建列表时就添加 3 个元素。然后使用 log 关键字将这个列表中的元素全部输出，如图 1-2-9 所示。

```
@{list1}    Create List hello    robot    framework
log ${list1}
```

1	@{list1}	Create List	hello	robot	framework
2	log	${list1}			

图 1-2-9

执行结果如图 1-2-10 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase2
20170305 22:53:44.087 :  INFO : @{list1} = [ hello | robot | framework ]
20170305 22:53:44.089 :  INFO : [u'hello', u'robot', u'framework']
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase2
```

图 1-2-10

1.2.7 如何定义一个字典

此处我们说的字典其实就等同于 Python 语言中的字典，和列表一样，字典也是 Python 语言中非常常用的一种数据结构，也类似于 Java 语言中的 Map。

在 Robot Framework 中，使用 Create Dictionary 来创建一个字典，比如我们定义一个字典 Dict1，并且在创建字典时就添加两个键值对，然后使用 Log Many 关键字将这个字典中的内容全部输出，如图 1-2-11 所示。

Log Many 关键字类似于 log 关键字，不同的是 log 关键字只可以接收一个参数，而 Log Many 关键字可以同时接收多个参数。

```
&{Dict1}     Create Dictionary  a=hello b=robotframework
Log Many    &{Dict1}
```

1	&{Dict1}	Create Dictionary	a=hello	b=robotframework
2	Log Many	&{Dict1}		

图 1-2-11

执行结果如图 1-2-12 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase003
20170305 23:23:49.439 :  INFO : &{Dict1} = { a=hello | b=robotframework }
20170305 23:23:49.439 :  INFO : a=hello
20170305 23:23:49.439 :  INFO : b=robotframework
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase003
```

图 1-2-12

1.2.8 如何拼接两个字符串

我们可以通过 Catenate 来拼接字符串，比如将“Hello”和“Robot”这两个字符串拼接起来并且输出，如图 1-2-13 所示。

```
${val2} Catenate      Hello  Robot
log ${val2}
```

1	${val2}	Catenate	Hello	Robot
2	log	${val2}		

图 1-2-13

执行结果如图 1-2-14 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase004
20170305 23:31:25.665 :  INFO : ${val2} = Hello Robot
20170305 23:31:25.681 :  INFO : Hello Robot
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase004
```

图 1-2-14

1.2.9　如何使用 for 循环

不管在哪种编程语言中，for 循环都是必不可少的。在 Robot Framework 中，我们也可以使用 for 循环来做遍历处理。

我们可以用 for 循环对一个列表进行遍历，并且输出该列表中的每一个元素。例如，list2 中有 a、b、c、d 四个元素，循环遍历输出这些元素，如图 1-2-15 所示。

```
@{list2}    Create List a   b   c   d
:FOR    ${value}     in @{list2}
    log ${value}
```

@{list2}	Create List	a	b	c	d
:FOR	${value}	in	@{list2}		
	log	${value}			

图 1-2-15

执行结果如图 1-2-16 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase005
20170307 23:18:13.430 :  INFO : @{list2} = [ a | b | c | d ]
20170307 23:18:13.430 :  INFO : a
20170307 23:18:13.430 :  INFO : b
20170307 23:18:13.430 :  INFO : c
20170307 23:18:13.430 :  INFO : d
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase005
```

图 1-2-16

1.2.10　如何中断 for 循环

我们可以使用 Exit For Loop If 关键字来中断一个 for 循环。例如，list2 有 a、b、c、d 四个元素，循环遍历输出这些元素，当输出到元素 c 时跳出这个循环，如图 1-2-17 所示。

```
@{list2}    Create List   a   b   c   d
:FOR     ${value}   in   @{list2}
    log  ${value}
    Exit For Loop If    '${value}'=='c'
```

1	@{list2}	Create List	a	b	c	d
2	:FOR	${value}	in	@{list2}		
3		log	${value}			
4		Exit For Loop If	'${value}'=='c'			

图 1-2-17

执行结果如图 1-2-18 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase006
20170307 23:26:50.877 :  INFO : @{list2} = [ a | b | c | d ]
20170307 23:26:50.877 :  INFO : a
20170307 23:26:50.893 :  INFO : b
20170307 23:26:50.893 :  INFO : c
20170307 23:26:50.893 :  INFO : Exiting for loop altogether.
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase006
```

图 1-2-18

1.2.11 Run Keyword If 判断的使用

Run Keyword If 是一个常用的用来做逻辑判断的关键字，意思是如果满足了某一个判断条件，就会执行关键字。我们在 list3 中放入 0、1、2 三个元素，然后遍历 list3，判断当取到元素 0 时，输出“男生”，如图 1-2-19 所示。

```
@{list3}    Create List 0    1    2
:FOR    ${value}    in  @{list3}
    Run Keyword If  '${value}'= ='0'    log 男生
...
```

1	@{list3}	Create List	0	1	2
2	:FOR	${value}	in	@{list3}	
3		Run Keyword If	'${value}'=='0'	log	男生
4	...				

图 1-2-19

执行结果如图 1-2-20 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase007
20170308 00:00:18.569 :  INFO : @{list3} = [ 0 | 1 | 2 ]
20170308 00:00:18.569 :  INFO : 男生
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase007
```

图 1-2-20

1.2.12 Comment 关键字的使用

Comment 关键字是用来做注释使用的，和很多编程语言中的注释作用一样，可以用来临时注释掉某一行自动化脚本，让其暂时不运行，也可以用来做解释说明使用，如图 1-2-21 所示。

在 Robot Framework 的 RIDE 中，可以选中某一行脚本，右击鼠标键，选择 Comment Rows

选项，然后可以对选中的那一行脚本做注释，注释完成后，这一行脚本将不会再被运行。

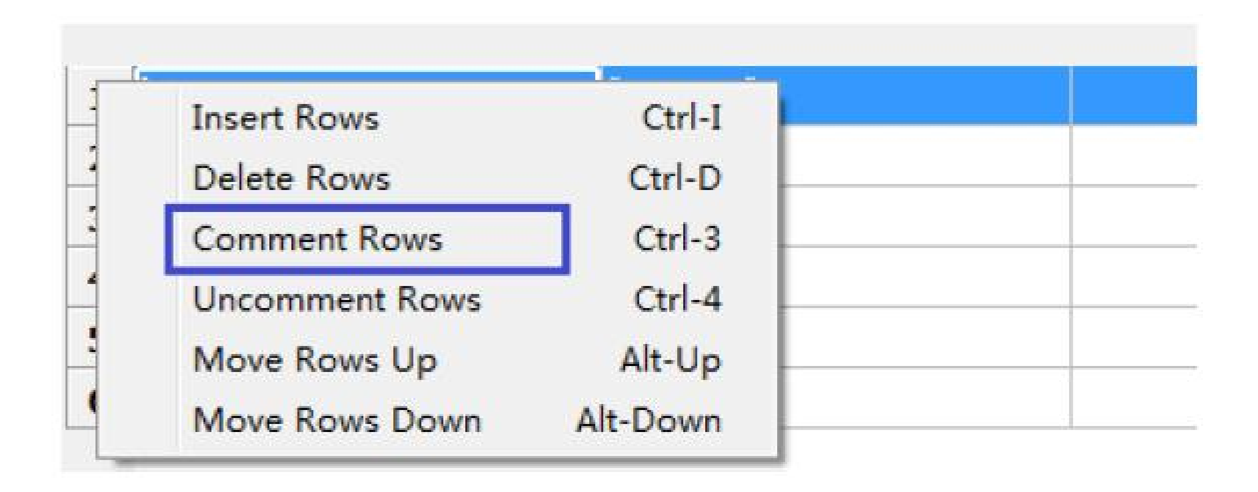


图 1-2-21

```
Comment      log "Comment"
```

如果需要取消注释，右击鼠标键，选择 Uncomment Rows 选项即可。注释取消后，在用例运行时，没有被注释的脚本就会被运行，如图 1-2-22 所示。

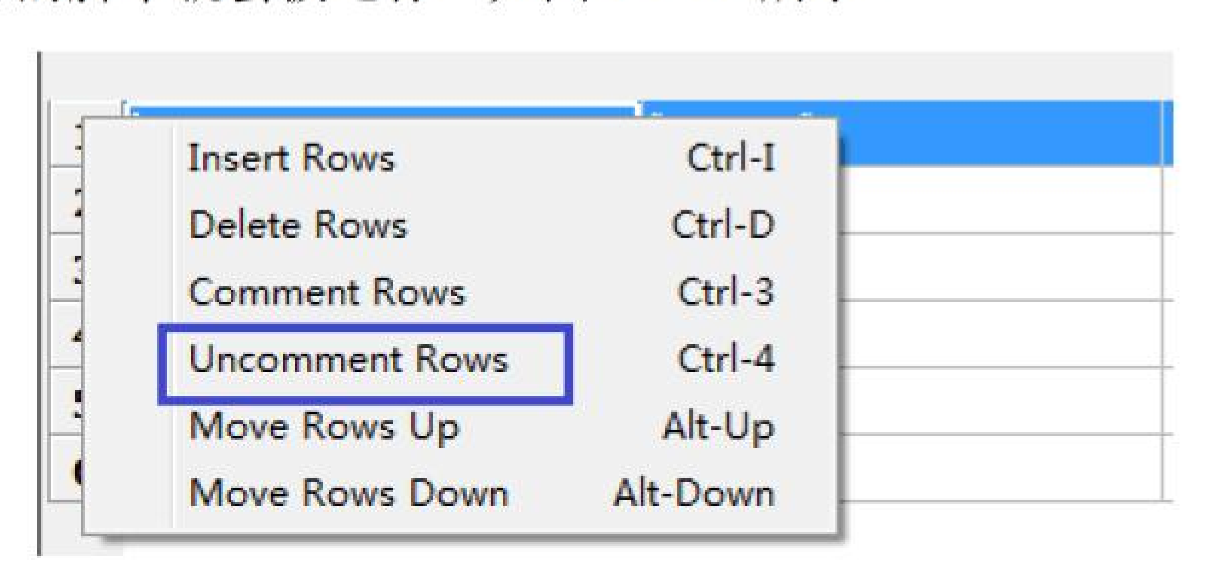


图 1-2-22

1.2.13　Return From Keyword 和 Return From Keyword If 关键字的使用

Return From Keyword 关键字和很多编程语言中的 return 关键字一样，具有如下鲜明的特点：

（1）脚本执行到该关键字后，会直接返回，不会再执行后面的脚本。

（2）返回时会带有对应返回值。调用者可以通过不同的返回值来建立不同的判断分支。

（3）Return From Keyword 关键字一般用于用户自定义关键字中。用户自定义关键字相当于是用系统已有的关键字来封装出一个新的关键字。

（4）Return From Keyword If 关键字用 if 条件来进行判断，当满足指定的 if 条件后，就执行 return 返回。返回时和 Return From Keyword 关键字一样，可以指定返回的具体值。

【示例】我们编写了一个自定义关键字，其中定义了一个入参${valueReturn}，如图 1-2-23 所示，然后通过执行 Return From Keyword If '${value}'=='${valueReturn}' ${value} 来判断我们需要返回的值，如图 1-2-24 所示。

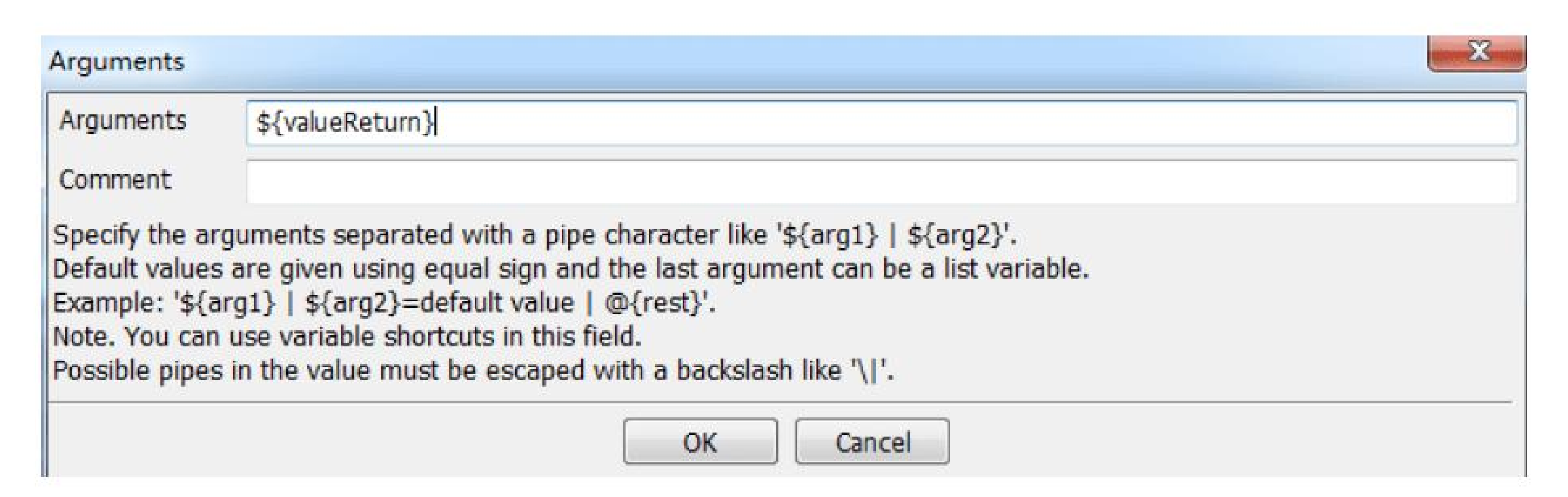


图 1-2-23

```
@{list2}    Create List 1    2    3    4
:FOR    ${value}    IN  @{list2}
    Return From Keyword If    '${value}'=='${valueReturn}'    ${value}
Return From Keyword    ${value}
```

Example Return From Keyword

Settings >>

1	@{list2}	Create List	1	2	3	4
2	: FOR	${value}	IN	@{list2}		
3		Return From Keyword If	'${value}'=='${valueReturn}'	${value}		
4	Return From Keyword	${value}				

图 1-2-24

自定义关键字完成后，就可以调用了，如图 1-2-25 所示。



${result}	Example Return From Keyword	4
log	${result}	

图 1-2-25

执行结果如图 1-2-26 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase030
20180924 11:44:12.210 :  INFO : @{list2} = [ 1 | 2 | 3 | 4 ]
20180924 11:44:12.219 :  INFO : Returning from the enclosing user keyword.
20180924 11:44:12.221 :  INFO : ${result} = 4
20180924 11:44:12.222 :  INFO : 4
Ending test:  RobotFrameworkTest1.TestSuite1.TestCase030
```

图 1-2-26

从执行结果可以看到，在调用 Example Return From Keyword 这个自定义关键字时，我们传入的入参为 4，按照自定义关键字中的判断逻辑返回 4。

1.3 Robot Framework 断言关键字

1.3.1　Should Be Equal 关键字的使用

Should Be Equal 关键字一般用来判断实际结果是否和预期结果相等。例如，我们将变量${value}的值设置为 1，使用 Should Be Equal 关键字来判断${value}是否等于 2，若断言失败，则输出实际值为${value}，和预期不符合，如图 1-3-1 所示。

```
${value}    Set Variable    1
Should Be Equal${value}    2    实际值为${value}，和预期不符合
```

${value}	Set Variable	1	
Should Be Equal	${value}	2	实际值为${value}，和预期不符合

图 1-3-1

执行结果如图 1-3-2 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase011
20170402 13:23:43.228 :  INFO : ${value} = 1
20170402 13:23:43.228 :  FAIL : 实际值为1, 和预期不符合: 1 != 2
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase011
```

图 1-3-2

1.3.2　Should Be True 关键字的使用

Should Be True 关键字用来判断返回值是否为 True，例如我们将变量${value}的值同样设置为 1，使用 Should Be True 关键字对表达式'${value}'=='2' 进行 True 和 False 的判断，由于我们设置的值为 1，因此很明显会判断失败，如图 1-3-3 所示。

```
${value}    Set Variable    1
Should Be True  '${value}'=='2' 判断失败
```

${value}	Set Variable	1
Should Be True	'${value}'=='2'	判断失败

图 1-3-3

执行结果如图 1-3-4 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase012
20170402 13:30:58.719 :  INFO : ${value} = 1
20170402 13:30:58.719 :  FAIL : 判断失败
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase012
```

图 1-3-4

1.3.3 Should Contain 关键字的使用

Should Contain 关键字用来判断某个字符串中是否包含了我们预期需要的字符或者字符串，例如我们将变量${str}的值设置为Robot Framework，使用Should Contain关键字来判断${str}是否包含“Hello”这个字符串。很明显，我们执行的结果肯定会判断失败，如图 1-3-5 所示。

```
${str}  Set Variable    RobotFramework
Should Contain ${str}  Hello    字符串${str}中不包含Hello
```

${str}	Set Variable	RobotFramework	
Should Contain	${str}	Hello	字符串${str}中不包含Hello

图 1-3-5

执行结果如图 1-3-6 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase013
20170402 13:44:52.194 :  INFO : ${str} = RobotFramework
20170402 13:44:52.194 :  FAIL : 字符串RobotFramework中不包含Hello: 'RobotFramework' does not contain 'Hello'
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase013
```

图 1-3-6

1.3.4 Should End With 关键字的使用

Should End With 关键字用来判断某个字符串是否以我们预期指定的字符串来结束，例如我们同样将变量${str}的值设置为 RobotFramework，使用 Should End With 来判断${str}是不是以“Hello”这个字符串来结束。很明显，我们执行的结果肯定会失败，如图 1-3-7 所示。

```
${str}  Set Variable    RobotFramework
Should End With${str}  Hello    字符串${str}中不以 Hello 来结束
```

${str}	Set Variable	RobotFramework	
Should End With	${str}	Hello	字符串${str}中不以Hello来结束

图 1-3-7

执行结果如图 1-3-8 所示。

```
Starting test: RobotFrameworkTest1.TestSuite1.TestCase014
20170402 13:39:15.737 :  INFO : ${str} = RobotFramework
20170402 13:39:15.737 :  FAIL : 字符串RobotFramework中不以Hello来结束: 'RobotFramework' does not end with 'Hello'
Ending test:   RobotFrameworkTest1.TestSuite1.TestCase014
```

图 1-3-8

当我们将“Hello”字符串换成“work”后，再执行一下，会发现执行成功，因为RobotFramework 是以 work 来结尾的，如图 1-3-9 所示。

```
${str}  Set Variable    RobotFramework
Should End With${str}  work    字符串${str}中不以 Hello 来结束
```

${str}	Set Variable	RobotFramework	
Should End With	${str}	work	字符串${str}中不以Hello来结束

图 1-3-9

执行结果如图 1-3-10 所示。

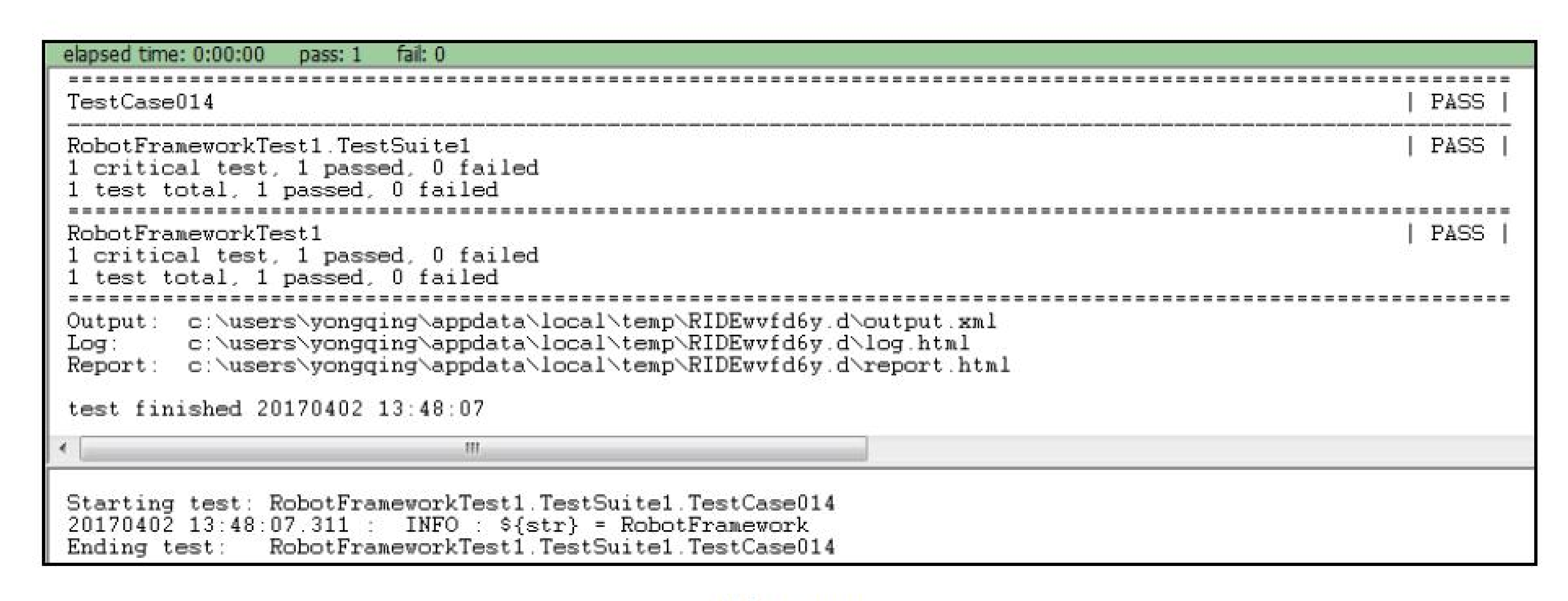


图 1-3-10

1.3.5 其他常用断言关键字

除了我们上面列出的关键字外，Robot Framework 中还提供了大量其他的断言关键字，如表 1-3-1 所示。

表 1-3-1 其他常用断言关键字

<table>
<tr><th>断言关键字</th><th>描述</th></tr>
<tr><td>Should Be Empty</td><td>判断是否为空，若不为空，则执行失败，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>Hello</td></tr><tr><td>Should Be Empty</td><td>${value}</td><td>结果不为空</td></tr></table></td></tr>
<tr><td>Should Start With</td><td>判断某个字符串是否以预期执行的字符串开始，若以指定的字符串开头，则执行成功，否则执行失败，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>Hello</td><td></td></tr><tr><td>Should Start With</td><td>${value}</td><td>qq</td><td>字符串${value}不以 qq 开头</td></tr></table></td></tr>
<tr><td>Should Not Start With</td><td>与 Should Start With 刚好相反，若以指定的字符串开头，则执行失败，否则执行成功，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>Hello</td><td></td></tr><tr><td>Should Not Start With</td><td>${value}</td><td>qq</td><td>字符串${value}是以 qq 开头</td></tr></table></td></tr>
<tr><td>Should Match</td><td>判断某个字符串是否与预期指定的字符串相匹配，若可以匹配，则执行成功，否则执行失败，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>Hello</td><td></td></tr><tr><td>Should Match</td><td>${value}</td><td>qq</td><td>字符串${value}不可以匹配 qq</td></tr></table></td></tr>
</table>

（续表）

<table>
<tr><th>断言关键字</th><th>描述</th></tr>
<tr><td>Should Not Match</td><td>与 Should Match 刚好相反，若字符串匹配，则执行失败，否则执行成功，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>Hello</td><td></td></tr>
<tr><td>Should Match</td><td>${value}</td><td>Hello</td><td>字 符 串 ${value} 可 以 匹 配 hello</td></tr>
</table></td></tr>
<tr><td>Should Contain X Times</td><td>与 Should Contain 关键字类似,用来判断指定的字符串包含指定的字符或者字符串多少次，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>hello</td><td></td><td></td></tr>
<tr><td>Should Contain X Times</td><td>${value}</td><td>hello</td><td>3</td><td>${value}中没有 3 次包含字符串 hello</td></tr>
</table></td></tr>
<tr><td>Should Be Equal As Integers</td><td>以整数的形式来进行比较，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>12</td><td></td></tr>
<tr><td>Should Be Equal As Integers</td><td>${value}</td><td>13</td><td>12 和 13 不相等</td></tr>
</table></td></tr>
<tr><td>Should Be Equal As Strings</td><td>以字符串的形式来进行比较，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>q</td><td></td></tr>
<tr><td>Should Be Equal As Strings</td><td>${value}</td><td>13</td><td>q 和 13 不相等</td></tr>
</table></td></tr>
<tr><td>Should Be Equal As Numbers</td><td>以 number 的形式来进行比较，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>1.0</td><td></td></tr>
<tr><td>Should Be Equal As Numbers</td><td>${value}</td><td>1</td><td>1.0 等于 1</td></tr>
</table></td></tr>
<tr><td>Should Not Be Equal</td><td>与 Should Be Equal 用法相反，当带比较的两个值相等时，执行失败，否则执行成功，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>1.0</td><td></td></tr>
<tr><td>Should Not Be Equal</td><td>${value}</td><td>1.0</td><td>1.0 等于 1. 0</td></tr>
</table></td></tr>
<tr><td>Should Not Be Empty</td><td>与 Should Be Empty 用法相反，若为空，则执行失败，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>Hello</td></tr>
<tr><td>Should Not Be Empty</td><td>${value}</td><td>字符串${value}为空</td></tr>
</table></td></tr>
</table>

1.4 BuiltIn 库剩余关键字

1.4.1 常用转换类型关键字

Robot Framework 中提供了很多类型转换关键字，如表 1-4-1 所示。

表 1-4-1　常用转换类型关键字

<table>
<tr><th>转换类型关键字</th><th>描述</th></tr>
<tr><td>Convert To Binary</td><td>将指定的内容转换为二进制形式，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td></tr><tr><td>${newvalue}</td><td>Convert To Binary</td><td>${value}</td></tr><tr><td>Log</td><td>${newvalue}</td><td></td></tr></table></td></tr>
<tr><td>Convert To Boolean</td><td>将指定的内容转换为布尔类型，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td></tr><tr><td>${newvalue}</td><td>Convert To Boolean</td><td>${value}</td></tr><tr><td>Log</td><td>${newvalue}</td><td></td></tr></table></td></tr>
<tr><td>Convert To Bytes</td><td>将指定的内容转换为字节数，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td><td></td></tr><tr><td>${newvalue}</td><td>Convert To Bytes</td><td>${value}</td><td>int</td></tr><tr><td>Log</td><td>${newvalue}</td><td></td><td></td></tr></table></td></tr>
<tr><td>Convert To Hex</td><td>将指定的内容转换为十六进制形式，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td></tr><tr><td>${newvalue}</td><td>Convert To Hex</td><td>${value}</td></tr><tr><td>Log</td><td>${newvalue}</td><td></td></tr></table></td></tr>
<tr><td>Convert To Integer</td><td>将指定的内容转换为 Integer 形式，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td></tr><tr><td>${newvalue}</td><td>Convert To Integer</td><td>${value}</td></tr><tr><td>Log</td><td>${newvalue}</td><td></td></tr></table></td></tr>
<tr><td>Convert To Number</td><td>将指定的内容转换为 Number 形式，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td></tr><tr><td>${newvalue}</td><td>Convert To Number</td><td>${value}</td></tr><tr><td>Log</td><td>${newvalue}</td><td></td></tr></table></td></tr>
<tr><td>Convert To Octal</td><td>将指定的内容转换为八进制形式，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td></tr><tr><td>${newvalue}</td><td>Convert To Octal</td><td>${value}</td></tr><tr><td>Log</td><td>${newvalue}</td><td></td></tr></table></td></tr>
<tr><td>Convert To String</td><td>将指定的内容转换为字符串形式，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td></tr><tr><td>${newvalue}</td><td>Convert To String</td><td>${value}</td></tr><tr><td>Log</td><td>${newvalue}</td><td></td></tr></table></td></tr>
</table>

1.4.2　常用 Get 类型关键字

表 1-4-2 中列出了 Get 类型关键字的常用用法。

表 1-4-2 Get 类型关键字的常用用法

<table>
<tr><th>Get 类型关键字</th><th>描述</th></tr>
<tr><td>Get Count</td><td>获取某个字符串包含指定字符或者字符串的次数，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>hellohello</td><td></td></tr>
<tr><td>${count}</td><td>Get Count</td><td>${value}</td><td>hello</td></tr>
<tr><td>Log</td><td>${count}</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Get Length</td><td>获取指定字符串的长度，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>hellohello</td></tr>
<tr><td>${length}</td><td>Get Length</td><td>${value}</td></tr>
<tr><td>log</td><td>${length}</td><td></td></tr>
</table></td></tr>
<tr><td>Get Time</td><td>获取时间，示例：
<table>
<tr><td>${time}</td><td>Get Time</td><td>format=timestamp</td></tr>
<tr><td>log</td><td>${time}</td><td></td></tr>
</table></td></tr>
<tr><td>Get Variable Value</td><td>获取指定变量的值，示例：
<table>
<tr><td>${value}</td><td>Set Variable</td><td>12</td></tr>
<tr><td>${result}</td><td>Get Variable Value</td><td>${value}</td></tr>
<tr><td>log</td><td>${result}</td><td></td></tr>
</table></td></tr>
<tr><td>Get Variables</td><td>获取所有的环境变量，示例：
<table>
<tr><td>${vars}</td><td>Get Variables</td></tr>
<tr><td>log</td><td>${vars}</td></tr>
</table></td></tr>
</table>

1.4.3 常用 Import 类型关键字

表 1-4-3 中列出了 Import 类型关键字的常用用法。

表 1-4-3 Import 类型关键字的常用用法

<table>
<tr><th>Import 类型关键字</th><th>描述</th></tr>
<tr><td>Import Library</td><td>在用例中导入某个 Library 库，示例：
<table>
<tr><td>Import Library</td><td>DatabaseLibrary</td></tr>
</table></td></tr>
<tr><td>Import Resource</td><td>在用例中导入某个 Resource 文件，示例：
<table>
<tr><td>Import Resource</td><td>d:\\RUNNING.txt</td></tr>
</table></td></tr>
<tr><td>Import Variables</td><td>在用例中从文件导入变量，示例：
<table>
<tr><td>Import Variables</td><td>variables.py</td></tr>
</table></td></tr>
</table>

1.4.4 常用 Set 类型关键字

表 1-4-4 中列出了 Set 类型关键字的常用用法。

表 1-4-4　Set 类型关键字的常用用法

<table>
<tr><th>Set 类型关键字</th><th>描述</th></tr>
<tr><td>Set Log Level</td><td>设置日志级别，示例：<table><tr><td>Set Log Level</td><td>DEBUG</td></tr></table></td></tr>
<tr><td>Set Variable If</td><td>根据判断条件的结果来确定给某个变量赋值，示例：<table><tr><td>${value}</td><td>Set Variable If</td><td>'2'>'1'</td><td>0</td></tr><tr><td>log</td><td>${value}</td><td></td><td></td></tr></table></td></tr>
<tr><td>Set Global Variable</td><td>设置全局变量，使得该变量也可以在别的用例中使用，示例：<table><tr><td>Set Global Variable</td><td>${book}</td><td>robotframework</td></tr><tr><td>log</td><td>${book}</td><td></td></tr></table></td></tr>
</table>

1.4.5　常用 Run Keyword 类型关键字

表 1-4-5 中列出了 Run Keyword 类型关键字的常用用法。

表 1-4-5　Run Keyword 类型关键字的常用用法

<table>
<tr><th>Run Keyword 类型关键字</th><th>描述</th></tr>
<tr><td>Run Keyword</td><td>执行某个关键字，示例：<table><tr><td>Run Keyword</td><td>log</td><td>RobotFrameWork</td></tr></table></td></tr>
<tr><td>Run Keywords</td><td>执行多个关键字，示例：<table><tr><td>Run Keywords</td><td>Set Global Variable</td><td>${book}</td><td>robotframework</td><td>AND</td><td>log</td><td>${book}</td></tr></table></td></tr>
<tr><td>Run Keyword And Return</td><td>Run Keyword And Return 和上面介绍的 Return From Keyword 和 Return From Keyword If 这两个关键很类似，必须要包装在用户自定义关键字中使用，主要用于执行一个指定关键字并且返回结果，接收[name | *args]多个参数，示例：首先需要定义一个用户自定义关键字，自定义关键字的名称为 Example_RUN_Keyword_AND_RETURN，关键字里面的内容如下：<table><tr><td>Run Keyword And Return</td><td>log</td><td>robotframework</td></tr></table>之后新建一个案例，调用该定义用户关键字：<table><tr><td>Example_RUN_Keyword_AND_RETURN</td></tr></table></td></tr>
</table>

1.4.6　其他关键字

表 1-4-6 中列出了 BuiltIn 库中剩余其他关键字的用法。

表 1-4-6　BuiltIn 库中剩余其他关键字的用法

<table>
<tr><th>关键字</th><th>描述</th></tr>
<tr><td>Evaluate</td><td>调用 Python 中给定的表达式，并且返回结果，示例：<table><tr><td>${value1}</td><td>Evaluate</td><td>int(3)+int(4)</td></tr><tr><td>Log</td><td>${value1}</td><td></td></tr></table></td></tr>
<tr><td>Fail</td><td>指定某个测试用例在执行某个步骤时直接判定执行失败，示例：<table><tr><td>Fail</td><td>执行失败</td></tr></table></td></tr>
<tr><td>Sleep</td><td>按照指定的时间休眠等待，示例：<table><tr><td>Sleep</td><td>3s</td></tr></table></td></tr>
<tr><td>Variable Should Exist</td><td>判断某个变量是否存在，若变量存在，则执行成功，否则执行失败，示例：<table><tr><td>Variable Should Exist</td><td>${value}</td><td>变量不存在</td></tr></table></td></tr>
<tr><td>Variable Should not Exist</td><td>和 Variable Should Exist 用法相反，若变量存在，则执行失败，否则执行成功，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>12</td></tr><tr><td>Variable Should not Exist</td><td>${value}</td><td>变量存在</td></tr></table></td></tr>
<tr><td>Wait Until Keyword Succeeds</td><td>在等待的时间内，若关键字执行失败，则按照每隔指定的时间重新执行，若超出等待的时间还执行失败，则执行失败，示例：<table><tr><td>Wait Until Keyword Succeeds</td><td>2 min</td><td>5 sec</td><td>Variable Should Exist</td><td>${value}</td></tr></table></td></tr>
<tr><td>Pass Execution</td><td>使用 PASS 状态跳过当前的测试，示例：<table><tr><td>Pass Execution</td><td>Deprecated test.</td></tr></table></td></tr>
<tr><td>Replace Variables</td><td>变量替换，示例：<table><tr><td>${value}</td><td>Set Variable</td><td>hello</td></tr><tr><td>${result}</td><td>Replace Variables</td><td>${value}</td></tr><tr><td>Should Be Equal</td><td>${result}</td><td>hello</td></tr></table></td></tr>
</table>

第 2 章 Robot Framework 对数据库的操作

2.1 DatabaseLibrary 库的使用

在自动化过程中，我们经常需要连接不同的数据库，并且对数据库进行很多不同的操作。Robot Framework 提供了 DatabaseLibrary 库来操作数据库。我们可以按照官网中的说明来安装 DatabaseLibrary 库。在浏览器中访问 http://franz-see.github.io/Robotframework-Database-Library/ 页面，即可看到该库的相关安装说明和 API 介绍，如图 2-1-1 所示。

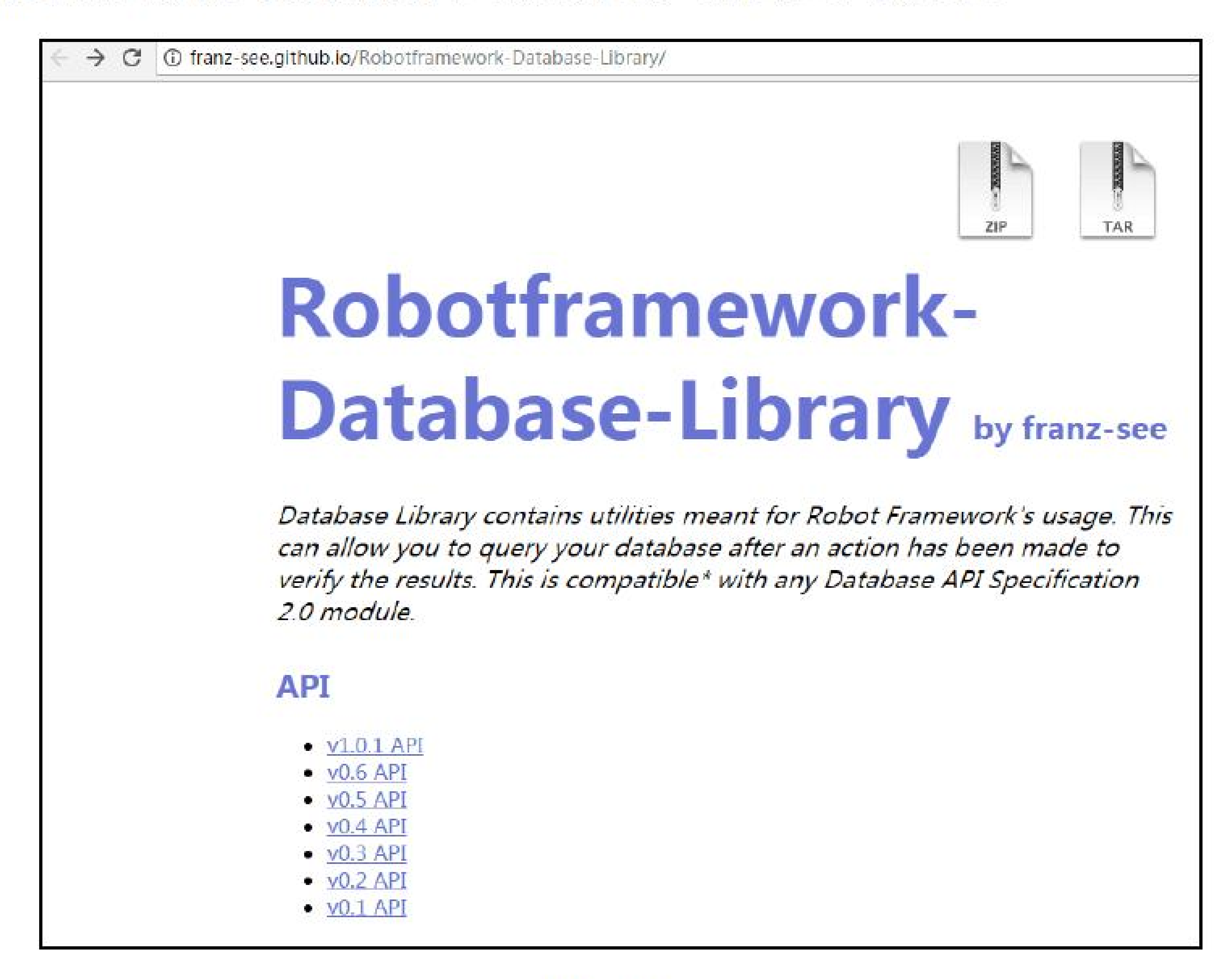


图 2-1-1

可以通过在命令行中执行 pip install -U robotframework-databaselibrary 来进行安装。安装完成后，在使用 DatabaseLibrary 库时，需要预先在测试套件中导入该库，如图 2-1-2 所示。这里以 MySQL 数据库为例，讲述 DatabaseLibrary 库的使用。

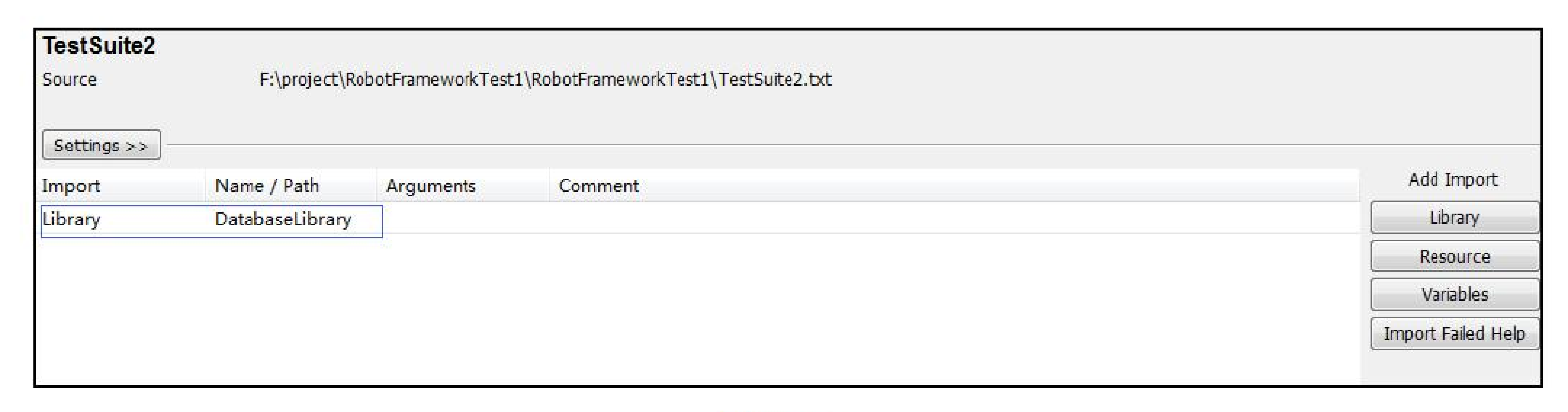


图 2-1-2

要连接到MySQL，除了要导入DatabaseLibrary库外，还需要安装pure-Python MySQL client library，可以通过访问网址：https://github.com/PyMySQL/PyMySQL 下载该库并且进行安装，或者直接在cmd命令行中输入pip install PyMySQL来进行安装，如图2-1-3所示。

```
C:\Users>pip install PyMySQL_
```

图 2-1-3

2.1.1 如何连接数据库

（1）可以通过DatabaseLibrary库中的Connect To Database关键字来连接一个MySQL数据库。此处以连接本机MySQL库为例，如图2-1-4所示。

- 数据库用户名：root。
- 数据库密码：root。
- MySQL数据库端口：3306。
- 数据库名：world。

```
Connect To Database    pymysql    world    root    root    localhost    3306
```

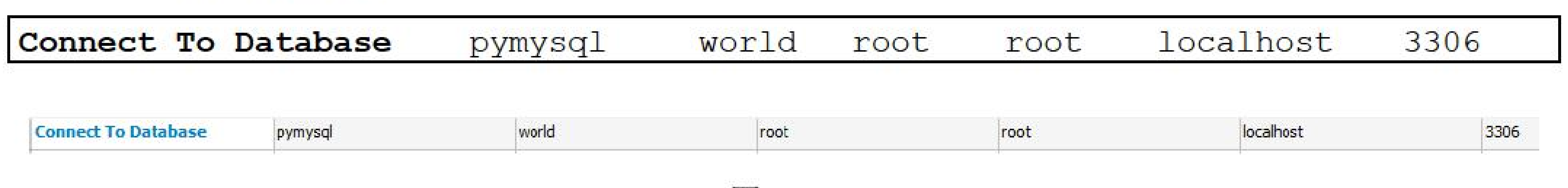


图 2-1-4

执行结果如图2-1-5所示。

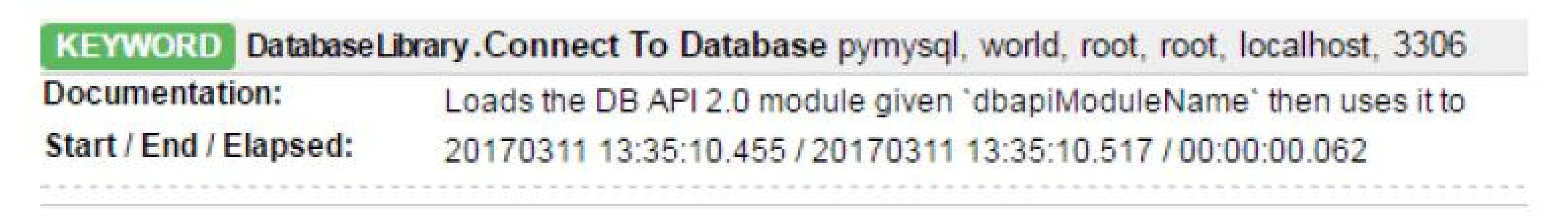
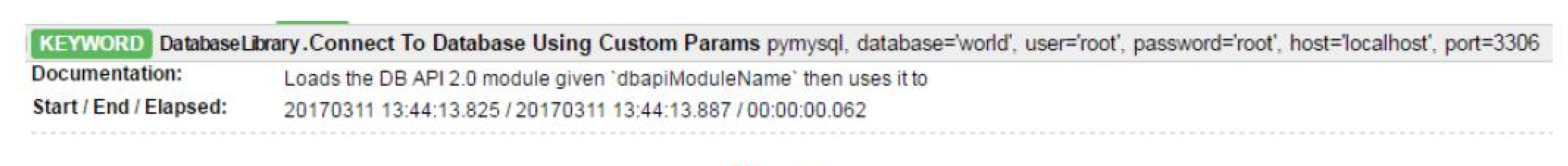


图 2-1-5

（2）还可以通过Connect To Database Using Custom Params关键字来连接MySQL数据库，如图2-1-6所示。

```
Connect To Database Using Custom Params    pymysql
database='world',user='root', password='root', host='localhost', port=3306
```



Connect To Database Using Custom Params	pymysql	database='world', user='root', password='root', host='localhost', port=3306

图 2-1-6

执行结果如图 2-1-7 所示。

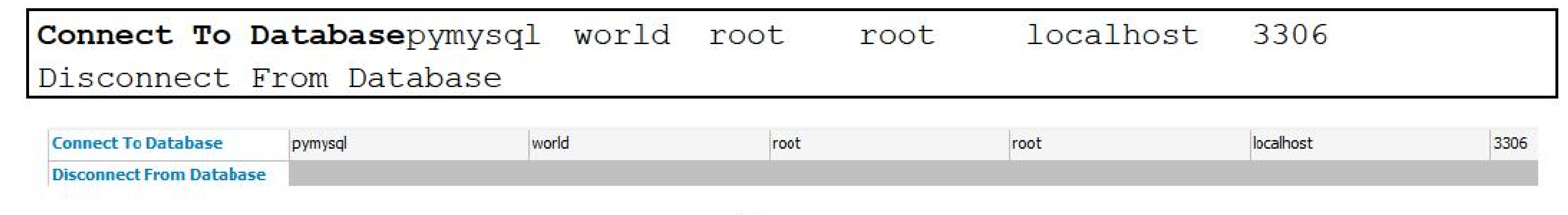


图 2-1-7

2.1.2　如何断开数据库

可以通过关键字 Disconnect From Database 断开数据库连接，我们在操作数据库时一定不要忘记在操作完成后断开数据库的连接，如图 2-1-8 所示。

```
Connect To Databasepymysql  world  root    root     localhost   3306
Disconnect From Database
```

Connect To Database	pymysql	world	root	root	localhost	3306
Disconnect From Database						

图 2-1-8

执行结果如图 2-1-9 所示。

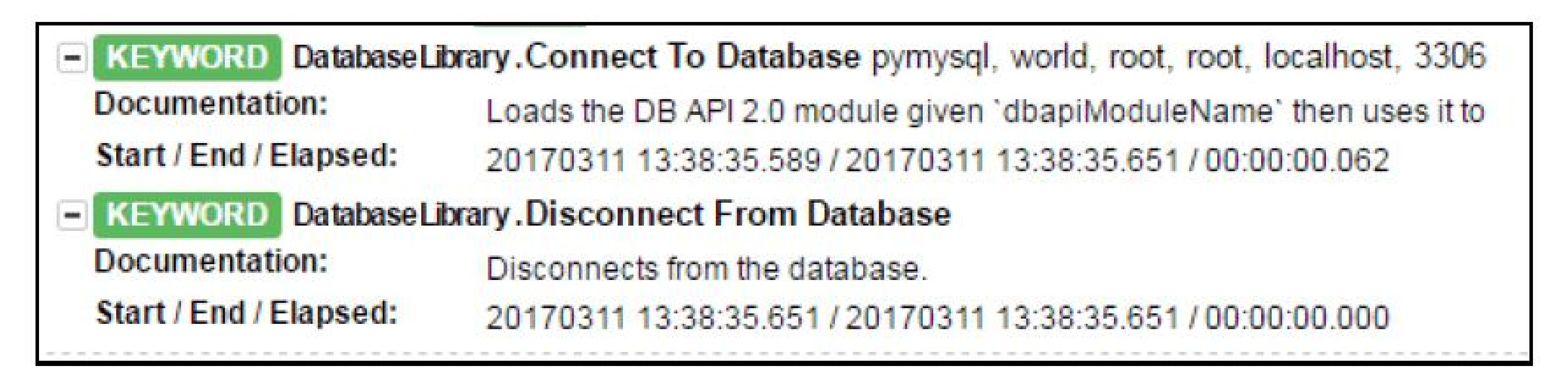
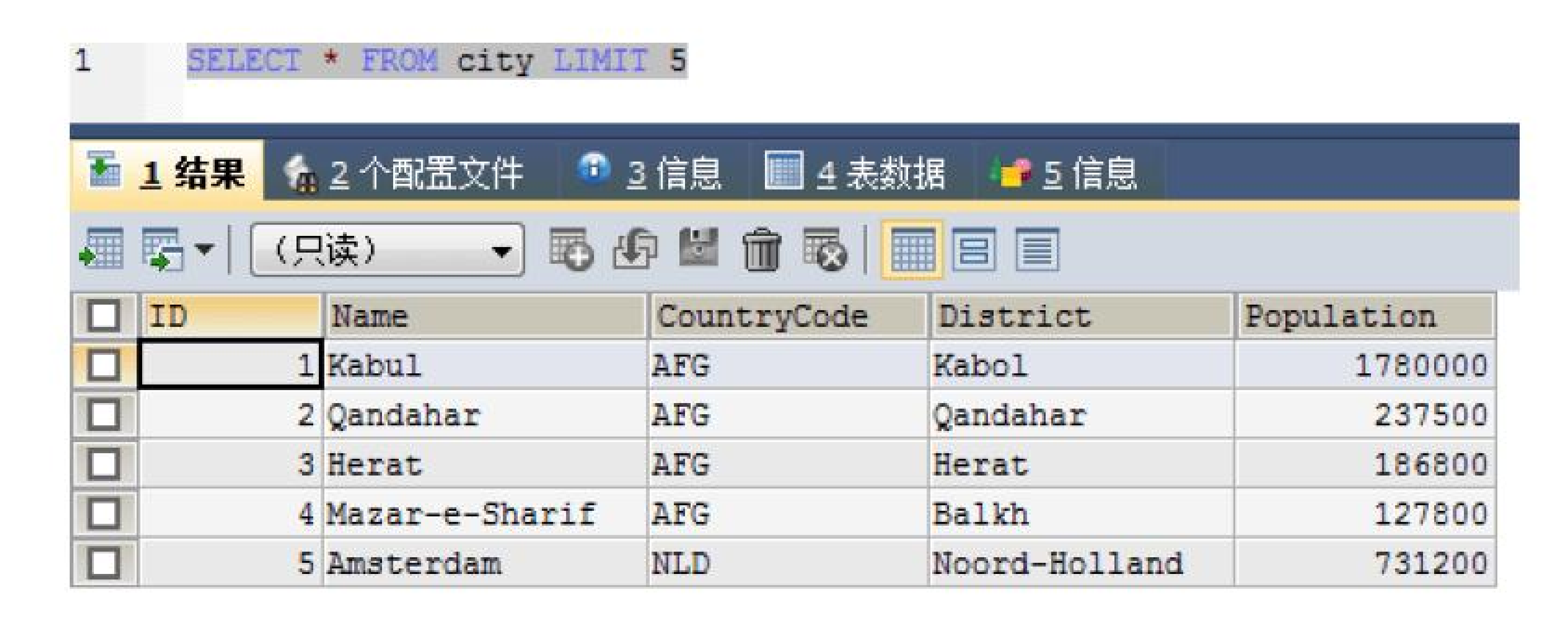


图 2-1-9

2.1.3　如何对数据库的表进行查询

通过 Query 关键字可以对数据库中的表进行查询。此处以查询 MySQL 数据库中某张表的数据为例，我们在 world 数据库中执行“SELECT * FROM city LIMIT 5;”这条 SQL 语句。在 SQL 窗口中查询出来的结果如图 2-1-10 所示。



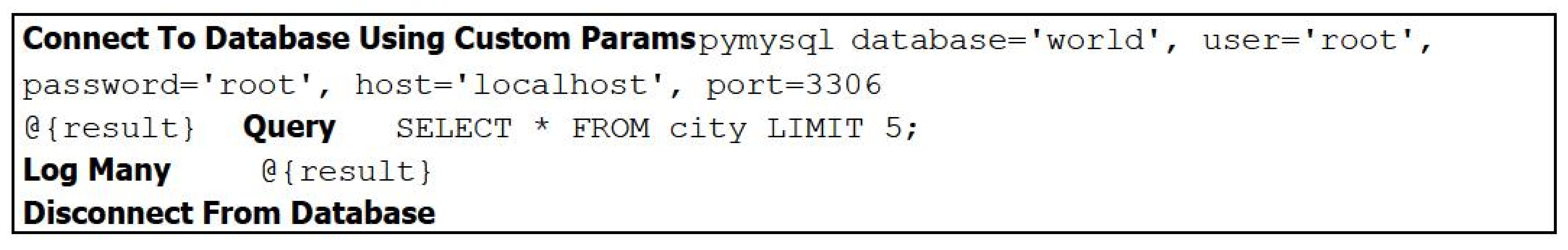
```
1    SELECT * FROM city LIMIT 5
```

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabol	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200

图 2-1-10

然后我们使用 Query 关键字来进行查询，如图 2-1-11 所示。

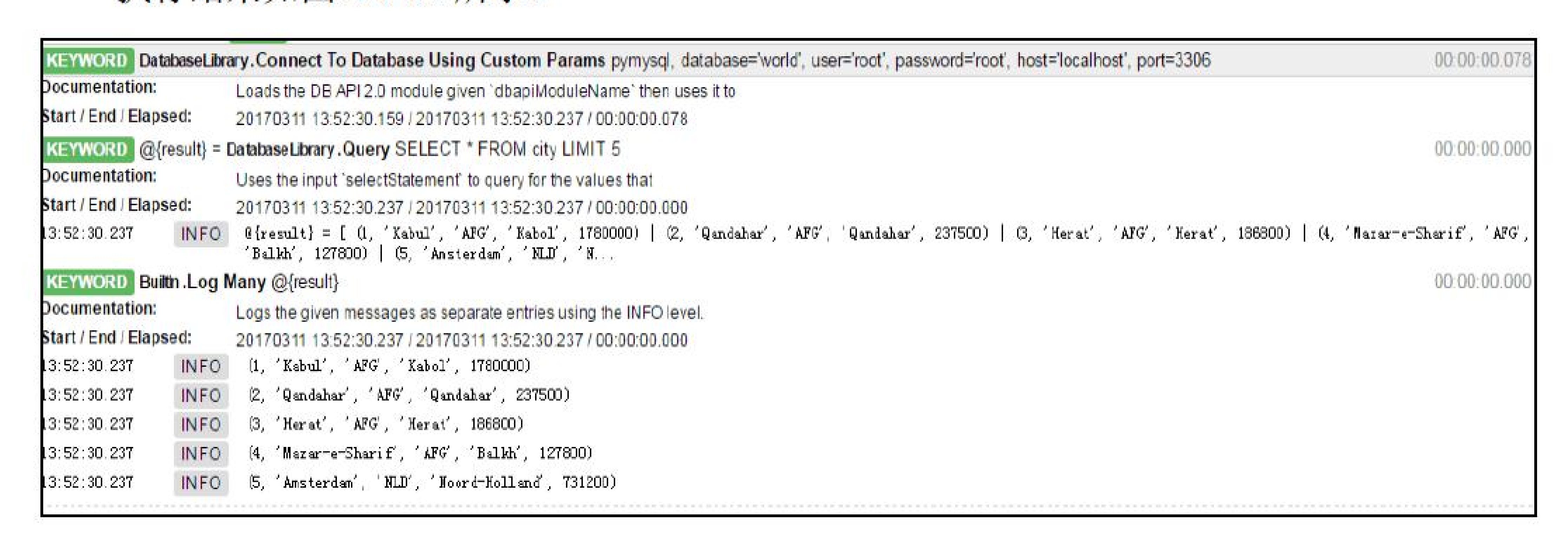
```
Connect To Database Using Custom Params pymysql database='world', user='root',
password='root', host='localhost', port=3306
@{result}   Query    SELECT * FROM city LIMIT 5;
Log Many      @{result}
Disconnect From Database
```



Connect To Database Using Custom Params	pymysql	database='world', user='root', password='root', host='localhost', port=3306
@{result}	Query	SELECT * FROM city LIMIT 5;
Log Many	@{result}	
Disconnect From Database		

图 2-1-11

执行结果如图 2-1-12 所示。



```
KEYWORD DatabaseLibrary.Connect To Database Using Custom Params pymysql, database='world', user='root', password='root', host='localhost', port=3306    00:00:00.078
Documentation:      Loads the DB API 2.0 module given `dbapiModuleName` then uses it to
Start / End / Elapsed:  20170311 13:52:30.159 / 20170311 13:52:30.237 / 00:00:00.078
KEYWORD @{result} = DatabaseLibrary.Query SELECT * FROM city LIMIT 5    00:00:00.000
Documentation:      Uses the input `selectStatement` to query for the values that
Start / End / Elapsed:  20170311 13:52:30.237 / 20170311 13:52:30.237 / 00:00:00.000
13:52:30.237  INFO  @{result} = [ (1, 'Kabul', 'AFG', 'Kabol', 1780000) | (2, 'Qandahar', 'AFG', 'Qandahar', 237500) | (3, 'Herat', 'AFG', 'Herat', 186800) | (4, 'Mazar-e-Sharif', 'AFG', 'Balkh', 127800) | (5, 'Amsterdam', 'NLD', 'N...
KEYWORD BuiltIn.Log Many @{result}    00:00:00.000
Documentation:      Logs the given messages as separate entries using the INFO level.
Start / End / Elapsed:  20170311 13:52:30.237 / 20170311 13:52:30.237 / 00:00:00.000
13:52:30.237  INFO  (1, 'Kabul', 'AFG', 'Kabol', 1780000)
13:52:30.237  INFO  (2, 'Qandahar', 'AFG', 'Qandahar', 237500)
13:52:30.237  INFO  (3, 'Herat', 'AFG', 'Herat', 186800)
13:52:30.237  INFO  (4, 'Mazar-e-Sharif', 'AFG', 'Balkh', 127800)
13:52:30.237  INFO  (5, 'Amsterdam', 'NLD', 'Noord-Holland', 731200)
```

图 2-1-12

2.1.4 如何插入和删除数据

可以通过 Execute Sql String 关键字来执行数据库的插入操作和删除操作。

（1）首先我们来看一下如何向数据库中插入数据。此处以向表 city 中插入一条记录为例，通过 Execute Sql String 关键字来执行 INSERT INTO city(NAME,countrycode,district,population)

VALUES('beijing' ,'ZH','China',217100) ，如图 2-1-13 所示。

```
Connect To Database Using Custom Params    pymysql database='world',
user='root', password='root', host='localhost', port=3306
Execute Sql String INSERT INTO city(NAME,countrycode,district,population)
VALUES('beijing' ,'ZH','China',217100)
Disconnect From Database
```

Connect To Database Using Custom Params	pymysql	database='world', user='root', password='root', host='localhost', port=3306
Execute Sql String	INSERT INTO city(NAME,countrycode,district,po VALUES('beijing' ,'ZH','China',217100)	
Disconnect From Database		

图 2-1-13

执行结果如图 2-1-14 所示。

KEYWORD DatabaseLibrary.Connect To Database Using Custom Params pymysql, database='world', user='root', password='root', host='localhost', port=3306
Documentation: Loads the DB API 2.0 module given `dbapiModuleName` then uses it to
Start / End / Elapsed: 20170311 14:11:04.375 / 20170311 14:11:04.422 / 00:00:00.047
KEYWORD DatabaseLibrary.Execute Sql String INSERT INTO city(NAME,countrycode,district,population) VALUES('beijing' ,'ZH','China',217100)
Documentation: Executes the sqlString as SQL commands.
Start / End / Elapsed: 20170311 14:11:04.422 / 20170311 14:11:04.422 / 00:00:00.000
KEYWORD DatabaseLibrary.Disconnect From Database
Documentation: Disconnects from the database.
Start / End / Elapsed: 20170311 14:11:04.422 / 20170311 14:11:04.422 / 00:00:00.000

图 2-1-14

在 SQL 窗口查询刚刚执行的 insert 语句是否执行成功。我们可以看到已经成功插入了数据，如图 2-1-15 所示。



图 2-1-15

（2）然后我们看一下怎么删除表中的数据。我们将上面插入的“'beijing' ,'ZH','China', 217100”这条数据从数据库中删除，如图 2-1-16 所示。

```
Connect To Database Using Custom Paramspymysql database='world', user='root',
password='root', host='localhost', port=3306
Execute Sql String delete from city where NAME='beijing'
Disconnect From Database
```

Connect To Database Using Custom Params	pymysql	database='world', user='root', password='root', host='localhost', port=3306
Execute Sql String	delete from city where NAME='beijing'	
Disconnect From Database		

图 2-1-16

执行结果如图 2-1-17 所示。

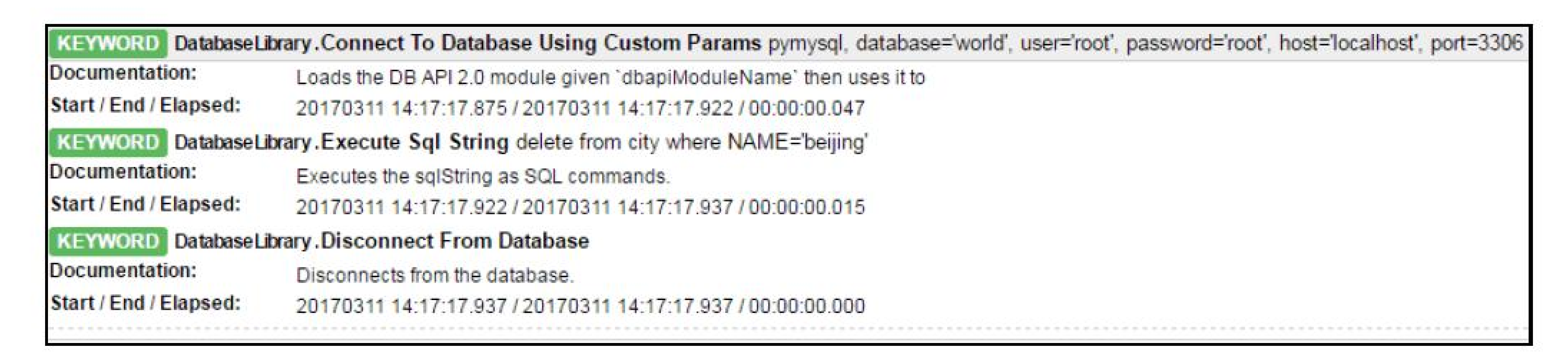

KEYWORD DatabaseLibrary.Connect To Database Using Custom Params pymysql, database='world', user='root', password='root', host='localhost', port=3306
Documentation: Loads the DB API 2.0 module given `dbapiModuleName` then uses it to
Start / End / Elapsed: 20170311 14:17:17.875 / 20170311 14:17:17.922 / 00:00:00.047
KEYWORD DatabaseLibrary.Execute Sql String delete from city where NAME='beijing'
Documentation: Executes the sqlString as SQL commands.
Start / End / Elapsed: 20170311 14:17:17.922 / 20170311 14:17:17.937 / 00:00:00.015
KEYWORD DatabaseLibrary.Disconnect From Database
Documentation: Disconnects from the database.
Start / End / Elapsed: 20170311 14:17:17.937 / 20170311 14:17:17.937 / 00:00:00.000

图 2-1-17

在 SQL 窗口查询一下有没有将数据成功删除。从查询的结果看，数据已经成功地被删除，如图 2-1-18 所示。

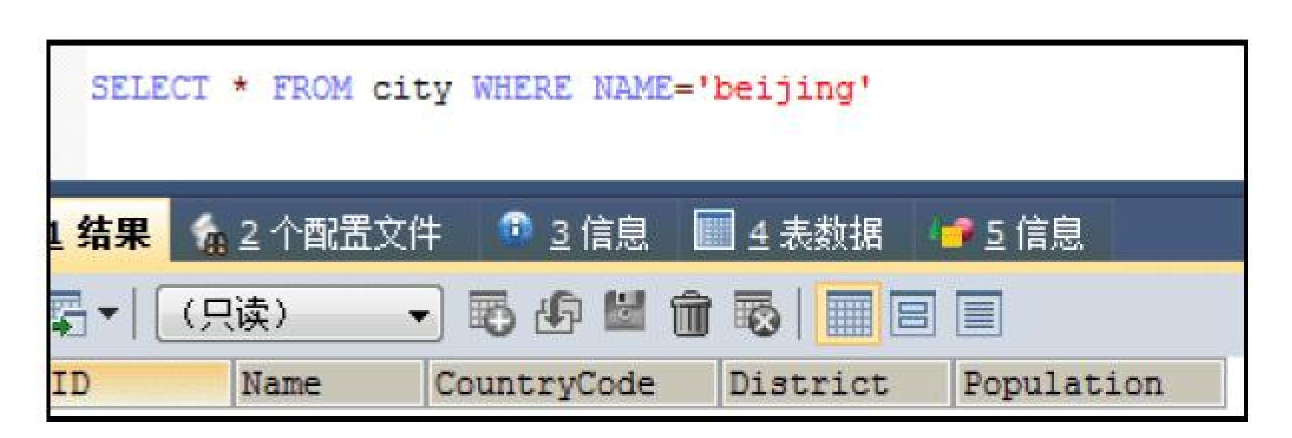


图 2-1-18

2.1.5 如何执行数据库脚本文件

在做自动化测试时，我们经常需要构造数据或者对库中的数据进行初始化，但是如果我们每次都是将要执行的数据库脚本按条写在用例中，那么将非常不好维护，因此我们需要直接执行数据库脚本文件。在 DatabaseLibrary 库中，可以通过 Execute Sql Script 关键字来执行数据库脚本文件。

此处以执行本地磁盘中的 script.sql 为例。在 script.sql 脚本中放入需要执行的语句，如图 2-1-19 所示。

script.sql - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
INSERT INTO city(NAME,countrycode,district,population) VALUES('beijing','ZH','China',217100);
INSERT INTO city(NAME,countrycode,district,population) VALUES('shanghai','ZH','China',226100);
```

图 2-1-19

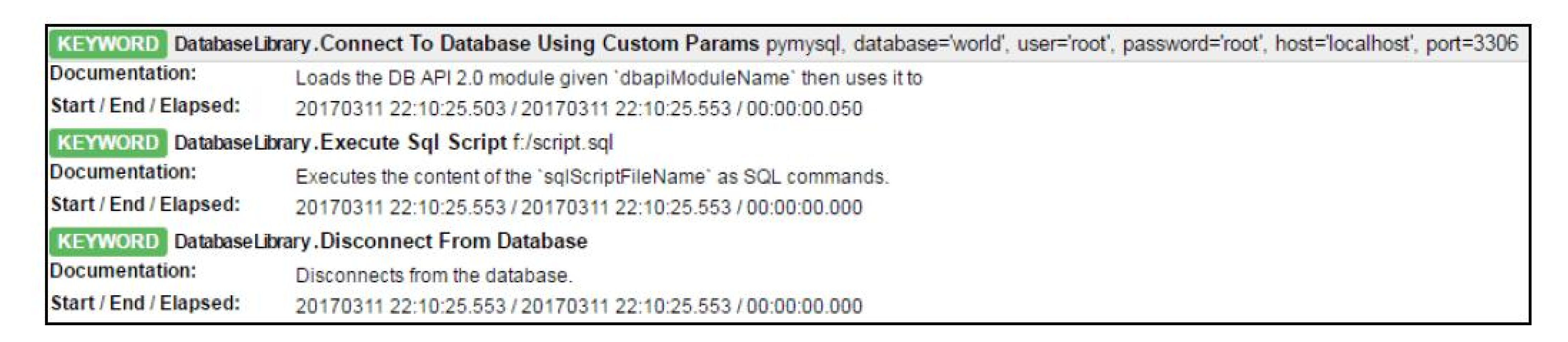
完整示例如图 2-1-20 所示。

```
Connect To Database Using Custom Params    pymysql
    database='world',user='root', password='root', host='localhost', port=3306
Execute Sql Script  f:/script.sql
Disconnect From Database
```

Connect To Database Using Custom Params	pymysql	database='world', user='root', password='root', host='localhost', port=3306
Execute Sql Script	f:/script.sql	
Disconnect From Database		

图 2-1-20

执行结果如图 2-1-21 所示。



KEYWORD DatabaseLibrary.Connect To Database Using Custom Params pymysql, database='world', user='root', password='root', host='localhost', port=3306
Documentation: Loads the DB API 2.0 module given `dbapiModuleName` then uses it to
Start / End / Elapsed: 20170311 22:10:25.503 / 20170311 22:10:25.553 / 00:00:00.050
KEYWORD DatabaseLibrary.Execute Sql Script f:/script.sql
Documentation: Executes the content of the `sqlScriptFileName` as SQL commands.
Start / End / Elapsed: 20170311 22:10:25.553 / 20170311 22:10:25.553 / 00:00:00.000
KEYWORD DatabaseLibrary.Disconnect From Database
Documentation: Disconnects from the database.
Start / End / Elapsed: 20170311 22:10:25.553 / 20170311 22:10:25.553 / 00:00:00.000

图 2-1-21

执行成功后，对数据库进行查询，会发现脚本已经执行成功、数据已经成功插入，如图 2-1-22 所示。



图 2-1-22

2.1.6　DatabaseLibrary 库的其他操作关键字

表 2-1-1 中描述了 DatabaseLibrary 库中其他关键字的使用方法。

表 2-1-1　DatabaseLibrary 库其他关键字的使用方法

关键字	使用描述
Check If Exists In Database	检查数据库查询是否有返回结果，若有返回结果，则用例执行成功，否则执行失败，示例： Check If Exists In Database ｜ SELECT * FROM city WHERE NAME='beijing' OR NAME='shanghai'

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Check If Not Exists In Database</td><td>检查数据库查询是否有返回结果，若有返回结果，则用例执行失败，否则执行成功，示例：<table><tr><td>Check If Not Exists In Database</td><td>SELECT * FROM city WHERE NAME='beijing' and NAME='shanghai'</td></tr></table></td></tr>
<tr><td>Delete All Rows From Table</td><td>删除数据库中某张表中的全部数据，示例：<table><tr><td>Delete All Rows From Table</td><td>World</td></tr></table></td></tr>
<tr><td>Description</td><td>描述数据库的查询结果，示例：<table><tr><td>@{result}</td><td>Description</td><td>SELECT * FROM city WHERE NAME='beijing' or NAME='shanghai'</td></tr><tr><td>Log Many</td><td>@{result}</td><td></td></tr></table></td></tr>
<tr><td>Row Count</td><td>统计 SQL 查询返回的记录数，示例：<table><tr><td>${rowCount}</td><td>Row Count</td><td>SELECT * FROM city WHERE NAME='beijing' or NAME='shanghai'</td></tr><tr><td>Log</td><td>${rowCount}</td><td></td></tr></table></td></tr>
<tr><td>Row Count Is 0</td><td>检查 SQL 查询返回的记录数是否为 0，示例：<table><tr><td>Row Count Is 0</td><td>SELECT * FROM city WHERE NAME='beijing' or NAME='shanghai'</td></tr></table></td></tr>
<tr><td>Row Count Is Equal To X</td><td>检查 SQL 查询返回的记录数是否等于某个值，示例：<table><tr><td>Row Count Is Equal To X</td><td>SELECT * FROM city WHERE NAME='beijing' or NAME='shanghai'</td><td>1</td></tr></table></td></tr>
<tr><td>Row Count Is Greater Than X</td><td>检查 SQL 查询返回的记录数是否大于某个值，示例：<table><tr><td>Row Count Is Greater Than X</td><td>SELECT * FROM city WHERE NAME='beijing' or NAME='shanghai'</td><td>1</td></tr></table></td></tr>
<tr><td>Row Count Is Less Than X</td><td>检查 SQL 查询返回的记录数是否小于某个值，示例：<table><tr><td>Row Count Is Less Than X</td><td>SELECT * FROM city WHERE NAME='beijing' or NAME='shanghai'</td><td>1</td></tr></table></td></tr>
<tr><td>Table Must Exist</td><td>判断数据库中表是否存在，示例：<table><tr><td>Table Must Exist</td><td>city</td></tr></table></td></tr>
</table>

2.2 MongoDBLibrary 库的使用

MongoDB 是非常常用的一个非关系型数据库。Robot Framework 提供了对 MongoDB 数据库测试操作的支持。我们可以通过在浏览器中访问 GitHub 的网站地址 https://github.com/iPlant

CollaborativeOpenSource/Robotframework-MongoDB-Library，查看该库的相关安装说明和 API 介绍，如图 2-2-1 所示。

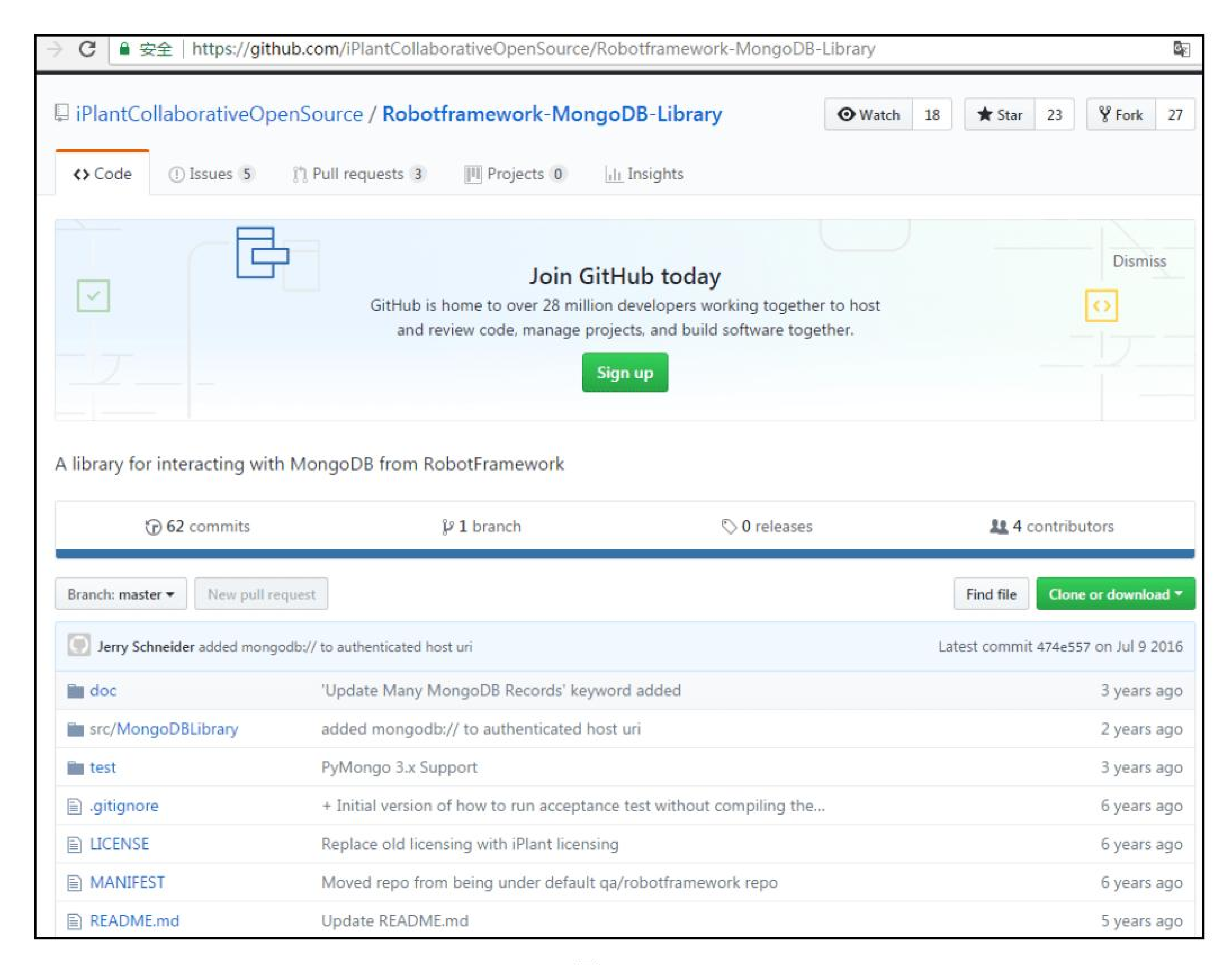


图 2-2-1

安装完成后，在使用 MongoDBLibrary 库时，需要预先在测试套件中导入该库，如图 2-2-2 所示。

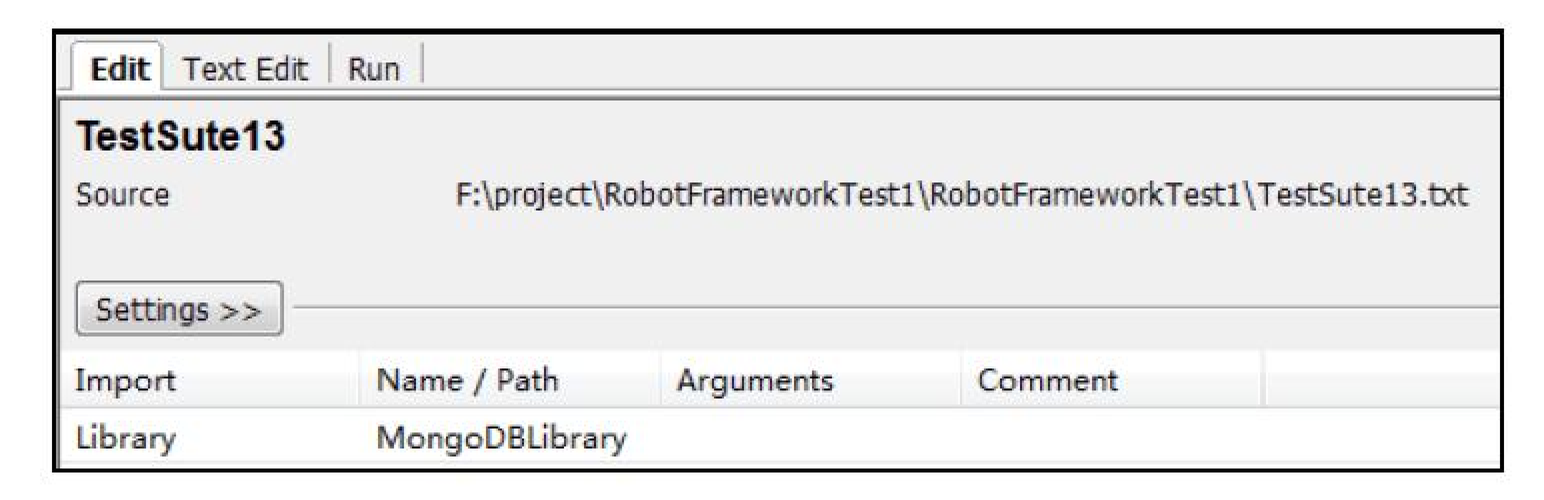


图 2-2-2

2.2.1　MongoDB 数据库的连接和断开

在 MongoDBLibrary 中通过 Connect To MongoDB 关键字来连接到 MongoDB 数据库，该

关键字接收[dbHost=localhost | dbPort=27017 | dbMaxPoolSize=10 | dbNetworkTimeout=None | dbDocClass= | dbTZAware=False]六个参数。其中，dbHost 参数指的是 MongoDB 数据库的 IP 地址，dbPort 参数指的是 MongoDB 数据库的端口号，不输入时默认为 27017；dbMaxPoolSize 参数指的是数据库连接的最大线程池大小，不输入时默认大小为 10。

【示例 1】我们连接到本地电脑上一个已经启动好的 MongoDB 数据库上，这里预先启动了一个 3.2 版本的 MongoDB 数据库，如图 2-2-3 所示。

```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\yongqing>cd D:\MongoDB\Server\3.2\bin

C:\Users\yongqing>d:

D:\MongoDB\Server\3.2\bin>mongod --dbpath "D:\MongoDB\Server\3.2\data" --logpath
 "D:\MongoDB\Server\3.2\logs"
2018-08-25T17:45:08.690+0800 F CONTROL  [main] Failed global initialization: Fil
eNotOpen: logpath "D:\MongoDB\Server\3.2\logs" should name a file, not a directo
ry.

D:\MongoDB\Server\3.2\bin>mongod --dbpath "D:\MongoDB\Server\3.2\data" --logpath
 "D:\MongoDB\Server\3.2\logs\log.log"
```

图 2-2-3

在 RIDE 中，使用 Connect To MongoDB 来连接刚刚启动好的数据库，如图 2-2-4 所示。

Connect To MongoDB	127.0.0.1	27017	2

图 2-2-4

运行结果如图 2-2-5 所示。

```
Starting test: RobotFrameworkTest1.TestSute13.TestCase001
20180825 17:47:32.470 :  INFO :
| Connect To MondoDB | dbHost | dbPort | dbMaxPoolSize | dbNetworktimeout |
dbDocClass | dbTZAware |
| Connect To MondoDB | 127.0.0.1 | 27017 | 2 | None | <type 'dict'> | False |
Ending test:  RobotFrameworkTest1.TestSute13.TestCase001
```

图 2-2-5

执行完成后，查看一下 MongoDB 服务端的日志。从如图 2-2-6 所示的 MongoDB 服务端的日志可以看到，已经成功和 MongoDB 数据库建立了连接。

```
2018-08-25T17:45:30.134+0800 I CONTROL  [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
2018-08-25T17:45:30.138+0800 I CONTROL  [initandlisten] MongoDB starting :
pid=8964 port=27017 dbpath=D:\MongoDB\Server\3.2\data 64-bit host=yongqing-PC
2018-08-25T17:45:30.138+0800 I CONTROL  [initandlisten] targetMinOS: Windows
Vista/Windows Server 2008
2018-08-25T17:45:30.139+0800 I CONTROL  [initandlisten] db version v3.2.4
2018-08-25T17:45:30.139+0800 I CONTROL  [initandlisten] git version:
e2ee9ffcf9f5a94fad76802e28cc978718bb7a30
2018-08-25T17:45:30.140+0800 I CONTROL  [initandlisten] allocator: tcmalloc
```

```
2018-08-25T17:45:30.140+0800 I CONTROL  [initandlisten] modules: none
2018-08-25T17:45:30.140+0800 I CONTROL  [initandlisten] build environment:
2018-08-25T17:45:30.161+0800 I CONTROL  [initandlisten]     distarch: x86_64
2018-08-25T17:45:30.162+0800 I CONTROL  [initandlisten]     target_arch: x86_64
2018-08-25T17:45:30.162+0800 I CONTROL  [initandlisten] options: { storage:
{ dbPath: "D:\MongoDB\Server\3.2\data" }, systemLog: { destination: "file", path:
"D:\MongoDB\Server\3.2\logs\log.log" } }
2018-08-25T17:45:30.164+0800 I STORAGE  [initandlisten] wiredtiger_open config:
create,cache_size=4G,session_max=20000,eviction=(threads_max=4),config_base=f
alse,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor
=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2
GB),statistics_log=(wait=0),
2018-08-25T17:45:31.202+0800 I FTDC     [initandlisten] Initializing full-time
diagnostic data capture with directory
'D:/MongoDB/Server/3.2/data/diagnostic.data'
2018-08-25T17:45:31.202+0800 I NETWORK  [HostnameCanonicalizationWorker]
Starting hostname canonicalization worker
2018-08-25T17:45:31.359+0800 I NETWORK  [initandlisten] waiting for connections
on port 27017
2018-08-25T17:46:53.205+0800 I NETWORK  [initandlisten] connection accepted from
127.0.0.1:2621 #1 (1 connection now open)
2018-08-25T17:46:53.730+0800 I NETWORK  [conn1] end connection 127.0.0.1:2621 (0
connections now open)
2018-08-25T17:47:32.471+0800 I NETWORK  [initandlisten] connection accepted from
127.0.0.1:2650 #2 (1 connection now open)
2018-08-25T17:47:33.031+0800 I NETWORK  [conn2] end connection 127.0.0.1:2650 (0
connections now open)
```

图 2-2-6

在 MongoDBLibrary 中通过 Disconnect From Mongodb 关键字来断开已经建立的 MongoDB 数据库连接。

【示例 2】通过 Disconnect From Mongodb 关键字断开 MongoDB 的数据库连接，如图 2-2-7 所示。

Connect To MongoDB	127.0.0.1	27017	2
Disconnect From Mongodb			

图 2-2-7

运行结果如图 2-2-8 所示。

```
Starting test: RobotFrameworkTest1.TestSute13.TestCase001
20180825 17:57:04.680 :  INFO :
| Connect To MondoDB | dbHost | dbPort | dbMaxPoolSize | dbNetworktimeout |
dbDocClass | dbTZAware |
| Connect To MondoDB | 127.0.0.1 | 27017 | 2 | None | <type 'dict'> | False |
20180825 17:57:04.684 :  INFO : | Disconnect From MongoDB |
Ending test:  RobotFrameworkTest1.TestSute13.TestCase001
```

图 2-2-8

从 MongoDB 的日志可以看到，在执行 Disconnect From Mongodb 关键字操作后，数据库服务端日志中已经显示数据库连接终止。

```
[conn3] end connection 127.0.0.1:2945 (0 connections now open)
```

2.2.2 Get Mongodb Databases 和 Get Mongodb Collections

在 MongoDBLibrary 中通过 Get Mongodb Databases 关键字来获取当前 MongoDB 下所有在用的数据库。

【示例 1】我们通过 Get Mongodb Databases 关键字来获取上面启动的 MongoDB 下的所有数据库，如图 2-2-9 所示。

Connect To MongoDB	127.0.0.1	27017	2
@{DBs}	Get Mongodb Databases		
Log Many	@{DBs}		
Disconnect From Mongodb			

图 2-2-9

运行结果如图 2-2-10 所示。

```
Starting test: RobotFrameworkTest1.TestSute13.TestCase003
20180825 21:55:34.867 :  INFO :
| Connect To MondoDB | dbHost | dbPort | dbMaxPoolSize | dbNetworktimeout |
dbDocClass | dbTZAware |
| Connect To MondoDB | 127.0.0.1 | 27017 | 2 | None | <type 'dict'> | False |
20180825 21:55:34.873 :  INFO : | @{allDBs} | Get Mongodb Databases |
20180825 21:55:34.873 :  INFO : @{DBs} = [ local ]
20180825 21:55:34.875 :  INFO : local
20180825 21:55:34.877 :  INFO : | Disconnect From MongoDB |
Ending test:  RobotFrameworkTest1.TestSute13.TestCase003
```

图 2-2-10

从运行结果可以看到只获取到了一个名叫 local 的数据库。我们通过客户端连接到 MongoDB 服务端，然后执行 show databases 命令，可以看到得到的结果和我们通过 Get Mongodb Databases 关键字来获取到的数据库信息是一致的，如图 2-2-11 所示。

```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:27017
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\yongqing>d:

D:\>cd D:\MongoDB\Server\3.2\bin

D:\MongoDB\Server\3.2\bin>mongo 127.0.0.1:27017
2018-08-25T21:57:46.261+0800 I CONTROL  [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
MongoDB shell version: 3.2.4
connecting to: 127.0.0.1:27017/test
> show databases;
local  0.000GB
>
```

图 2-2-11

通过 Get Mongodb Collections 关键字可以获取到指定 MongoDB 数据库下的所有 Collection，该关键字接收[dbName]一个参数。

【示例 2】我们通过 Get Mongodb Collections 关键字来获取到 local 库下的所有 Collection，如图 2-2-12 所示。

Connect To MongoDB	127.0.0.1	27017	2
@{DBs}	Get Mongodb Databases		
Log Many	@{DBs}		
@{allCollections}	Get Mongodb Collections	local	
Log Many	@{allCollections}		
Disconnect From Mongodb			

图 2-2-12

运行结果如图 2-2-13 所示。

```
Starting test: RobotFrameworkTest1.TestSute13.TestCase002
20180825 22:03:31.189 :  INFO :
| Connect To MondoDB | dbHost | dbPort | dbMaxPoolSize | dbNetworktimeout |
dbDocClass | dbTZAware |
| Connect To MondoDB | 127.0.0.1 | 27017 | 2 | None | <type 'dict'> | False |
20180825 22:03:31.197 :  INFO : | @{allDBs} | Get Mongodb Databases |
20180825 22:03:31.198 :  INFO : @{DBs} = [ local ]
20180825 22:03:31.200 :  INFO : local
20180825 22:03:31.205 :  INFO : | @{allCollections} | Get MongoDB Collections |
local |
20180825 22:03:31.206 :  INFO : @{allCollections} = [ startup_log ]
20180825 22:03:31.208 :  INFO : startup_log
20180825 22:03:31.210 :  INFO : | Disconnect From MongoDB |
Ending test:   RobotFrameworkTest1.TestSute13.TestCase002
```

图 2-2-13

从运行结果可以看到，获取到的 local 库下的 Collection 名叫 startup_log，然后我们通过客户端连接到服务端，通过客户端 show collections 命令来获取 Collection。我们可以看到获取到的 Collection 是完全一致的，如图 2-2-14 所示。

```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:27017
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\yongqing>d:

D:\>cd D:\MongoDB\Server\3.2\bin

D:\MongoDB\Server\3.2\bin>mongo 127.0.0.1:27017
2018-08-25T21:57:46.261+0800 I CONTROL  [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
MongoDB shell version: 3.2.4
connecting to: 127.0.0.1:27017/test
> show databases;
local  0.000GB
> use local
switched to db local
> show collections
startup_log
>
```

图 2-2-14

2.2.3 Save Mongodb Records

Save MongoDB Records 关键字用来向指定的 Collection 中保存插入的记录，接收[dbName | dbCollName | recordJSON]三个参数。

【示例】我们向 startup_log 这个 Collection 中插入一条记录，如图 2-2-15 所示。



Connect To MongoDB	127.0.0.1	27017	2
Save Mongodb Records	local	startup_log	{"book":"RobotFramework"}
Disconnect From Mongodb			

图 2-2-15

运行结果如图 2-2-16 所示。

```
Starting test: RobotFrameworkTest1.TestSute13.TestCase004
20180825 22:16:34.333 :  INFO :
| Connect To MondoDB | dbHost | dbPort | dbMaxPoolSize | dbNetworktimeout |
dbDocClass | dbTZAware |
| Connect To MondoDB | 127.0.0.1 | 27017 | 2 | None | <type 'dict'> | False |
20180825 22:16:34.344 :  INFO : | ${allResults} | Save MongoDB Records | local
| startup_log | {u'book': u'RobotFramework', '_id':
ObjectId('5b8164c2685b132ec4739503')} |
20180825 22:16:34.347 :  INFO : | Disconnect From MongoDB |
Ending test:  RobotFrameworkTest1.TestSute13.TestCase004
```

图 2-2-16

执行完成后，我们通过 MongoDB 客户端连接到服务端，执行 db.startup_log.find()命令来查看 startup_log 这个 Collection 下的记录。可以看到{"book":"RobotFramework"}这条数据记录已经成功插入 MongoDB 中，如图 2-2-17 所示。

```
管理员: C:\Windows\system32\cmd.exe - mongo  127.0.0.1:27017
connecting to: 127.0.0.1:27017/test
> use local
switched to db local
> db.startup_log.find();
{ "_id" : "yongqing-PC-1535190331203", "hostname" : "yongqing-PC", "startTime" :
 ISODate("2018-08-25T09:45:31Z"), "startTimeLocal" : "Sat Aug 25 17:45:31.203",
"cmdLine" : { "storage" : { "dbPath" : "D:\\MongoDB\\Server\\3.2\\data" }, "syst
emLog" : { "destination" : "file", "path" : "D:\\MongoDB\\Server\\3.2\\logs\\log
.log" } }, "pid" : NumberLong(8964), "buildinfo" : { "version" : "3.2.4", "gitVe
rsion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30", "targetMinOS" : "Windows Vi
sta/Windows Server 2008", "modules" : [ ], "allocator" : "tcmalloc", "javascript
Engine" : "mozjs", "sysInfo" : "deprecated", "versionArray" : [ 3, 2, 4, 0 ], "o
penssl" : { "running" : "disabled", "compiled" : "disabled" }, "buildEnvironment
" : { "distmod" : "", "distarch" : "x86_64", "cc" : "cl: Microsoft (R) C/C++ Opt
imizing Compiler Version 18.00.31101 for x64", "ccflags" : "/nologo /EHsc /W3 /w
d4355 /wd4800 /wd4267 /wd4244 /wd4290 /wd4068 /wd4351 /we4013 /we4099 /we4930 /Z
7 /errorReport:none /MT /O2 /Oy- /Gw /Gy /Zc:inline", "cxx" : "cl: Microsoft (R)
 C/C++ Optimizing Compiler Version 18.00.31101 for x64", "cxxflags" : "/TP", "li
nkflags" : "/nologo /DEBUG /INCREMENTAL:NO /LARGEADDRESSAWARE /OPT:REF", "target
_arch" : "x86_64", "target_os" : "windows" }, "bits" : 64, "debug" : false, "max
BsonObjectSize" : 16777216, "storageEngines" : [ "devnull", "ephemeralForTest",
"mmapv1", "wiredTiger" ] } }
{ "_id" : ObjectId("5b8164c2685b132ec4739503"), "book" : "RobotFramework" }
>
> _
```

图 2-2-17

2.2.4 Retrieve All Mongodb Records

Retrieve All Mongodb Records 关键字用来获取指定 Collection 下的所有数据记录，接收 [dbName | dbCollName | returnDocuments=False]三个参数。

【示例】我们通过 Retrieve All Mongodb Records 关键字来获取 startup_log 下的数据记录，如图 2-2-18 所示。

Connect To MongoDB	127.0.0.1	27017	2	
${allResults}	Retrieve All Mongodb Records	local	startup_log	True
log	${allResults}			
Disconnect From Mongodb				

图 2-2-18

运行结果如图 2-2-19 所示。

```
Starting test: RobotFrameworkTest1.TestSute13.TestCase005
20180825 22:38:05.197 :  INFO :
| Connect To MondoDB | dbHost | dbPort | dbMaxPoolSize | dbNetworktimeout |
dbDocClass | dbTZAware |
| Connect To MondoDB | 127.0.0.1 | 27017 | 2 | None | <type 'dict'> | False |
20180825 22:38:05.213 :  INFO : ${allResults} = [{u'hostname': u'yongqing-PC',
u'pid': 8964L, u'startTimeLocal': u'Sat Aug 25 17:45:31.203', u'cmdLine':
{u'storage': {u'dbPath': u'D:\\MongoDB\\Server\\3.2\\data'}, u'systemLog':
{u'path': u'D:\\Mong...
20180825 22:38:05.215 :  INFO : [{u'hostname': u'yongqing-PC', u'pid': 8964L,
u'startTimeLocal': u'Sat Aug 25 17:45:31.203', u'cmdLine': {u'storage':
{u'dbPath': u'D:\\MongoDB\\Server\\3.2\\data'}, u'systemLog': {u'path':
u'D:\\MongoDB\\Server\\3.2\\logs\\log.log', u'destination': u'file'}},
u'startTime': datetime.datetime(2018, 8, 25, 9, 45, 31), u'_id':
u'yongqing-PC-1535190331203', u'buildinfo': {u'storageEngines': [u'devnull',
u'ephemeralForTest', u'mmapv1', u'wiredTiger'], u'maxBsonObjectSize': 16777216,
u'bits': 64, u'sysInfo': u'deprecated', u'modules': [], u'openssl': {u'compiled':
u'disabled', u'running': u'disabled'}, u'javascriptEngine': u'mozjs',
u'version': u'3.2.4', u'gitVersion':
u'e2ee9ffcf9f5a94fad76802e28cc978718bb7a30', u'versionArray': [3, 2, 4, 0],
u'debug': False, u'buildEnvironment': {u'cxxflags': u'/TP', u'cc': u'cl:
Microsoft (R) C/C++ Optimizing Compiler Version 18.00.31101 for x64',
u'linkflags': u'/nologo /DEBUG /INCREMENTAL:NO /LARGEADDRESSAWARE /OPT:REF',
u'distarch': u'x86_64', u'cxx': u'cl: Microsoft (R) C/C++ Optimizing Compiler
Version 18.00.31101 for x64', u'ccflags': u'/nologo /EHsc /W3 /wd4355 /wd4800
/wd4267 /wd4244 /wd4290 /wd4068 /wd4351 /we4013 /we4099 /we4930 /Z7
/errorReport:none /MT /O2 /Oy- /Gw /Gy /Zc:inline', u'target_arch': u'x86_64',
u'distmod': u'', u'target_os': u'windows'}, u'targetMinOS': u'Windows
Vista/Windows Server 2008', u'allocator': u'tcmalloc'}}, {u'_id':
ObjectId('5b8164c2685b132ec4739503'), u'book': u'RobotFramework'}]
20180825 22:38:05.217 :  INFO : | Disconnect From MongoDB |
Ending test:   RobotFrameworkTest1.TestSute13.TestCase005
```

图 2-2-19

如图 2-2-20 所示，运行结果与我们在客户端通过 db.startup_log.find()命令行获取到的结果是一致的。

```
管理员: C:\Windows\system32\cmd.exe - mongo  127.0.0.1:27017
connecting to: 127.0.0.1:27017/test
> use local
switched to db local
> db.startup_log.find();
{ "_id" : "yongqing-PC-1535190331203", "hostname" : "yongqing-PC", "startTime" :
 ISODate("2018-08-25T09:45:31Z"), "startTimeLocal" : "Sat Aug 25 17:45:31.203",
"cmdLine" : { "storage" : { "dbPath" : "D:\\MongoDB\\Server\\3.2\\data" }, "syst
emLog" : { "destination" : "file", "path" : "D:\\MongoDB\\Server\\3.2\\logs\\log
.log" } }, "pid" : NumberLong(8964), "buildinfo" : { "version" : "3.2.4", "gitVe
rsion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30", "targetMinOS" : "Windows Vi
sta/Windows Server 2008", "modules" : [ ], "allocator" : "tcmalloc", "javascript
Engine" : "mozjs", "sysInfo" : "deprecated", "versionArray" : [ 3, 2, 4, 0 ], "o
penssl" : { "running" : "disabled", "compiled" : "disabled" }, "buildEnvironment
" : { "distmod" : "", "distarch" : "x86_64", "cc" : "cl: Microsoft (R) C/C++ Opt
imizing Compiler Version 18.00.31101 for x64", "ccflags" : "/nologo /EHsc /W3 /w
d4355 /wd4800 /wd4267 /wd4244 /wd4290 /wd4068 /wd4351 /we4013 /we4099 /we4930 /Z
7 /errorReport:none /MT /O2 /Oy- /Gw /Gy /Zc:inline", "cxx" : "cl: Microsoft (R)
 C/C++ Optimizing Compiler Version 18.00.31101 for x64", "cxxflags" : "/TP", "li
nkflags" : "/nologo /DEBUG /INCREMENTAL:NO /LARGEADDRESSAWARE /OPT:REF", "target
_arch" : "x86_64", "target_os" : "windows" }, "bits" : 64, "debug" : false, "max
BsonObjectSize" : 16777216, "storageEngines" : [ "devnull", "ephemeralForTest",
"mmapv1", "wiredTiger" ] } }
{ "_id" : ObjectId("5b8164c2685b132ec4739503"), "book" : "RobotFramework" }
>
>
```

图 2-2-20

2.2.5 Update Many Mongodb Records

Update Many Mongodb Records 关键字用来更新 Collection 中的数据记录，接收[dbName | dbCollName | queryJSON | updateJSON | upsert=False]五个参数。

【示例】我们更新上面示例中插入的{"book":"RobotFramework"}记录为{"book":"robotFramework"}，即将 RobotFramework 变为 robotFramework，如图 2-2-21 所示。

Connect To MongoDB	127.0.0.1	27017	2	
${newJson}	Set Variable	{"$set": {"book":"robotFramework"}}		
Update Many Mongodb Records	local	startup_log	{"book":"RobotFramework"}	${newJson}
Disconnect From Mongodb				

图 2-2-21

运行结果如图 2-2-22 所示。

```
Starting test: RobotFrameworkTest1.TestSute13.TestCase006
20180825 23:05:53.692 :  INFO :
| Connect To MondoDB | dbHost | dbPort | dbMaxPoolSize | dbNetworktimeout |
dbDocClass | dbTZAware |
| Connect To MondoDB | 127.0.0.1 | 27017 | 2 | None | <type 'dict'> | False |
20180825 23:05:53.700 :  INFO : ${newJson} = {"$set": {"book":"robotFramework"}}
20180825 23:05:53.708 :  INFO :
Matched: 1 documents
| ${allResults} | Update Many MongoDB Records | local | startup_log | {u'book':
u'RobotFramework'} | {u'$set': {u'book': u'robotFramework'}} |
20180825 23:05:53.710 :  INFO : | Disconnect From MongoDB |
Ending test:   RobotFrameworkTest1.TestSute13.TestCase006
```

图 2-2-22

更新完成后，通过客户端的 db.startup_log.find()命令来进行重新查询，如图 2-2-23 所示。从查询的结果可以看到指定的记录已经更新完成了。

```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:27017
_arch" : "x86_64", "target_os" : "windows" }, "bits" : 64, "debug" : false, "max
BsonObjectSize" : 16777216, "storageEngines" : [ "devnull", "ephemeralForTest",
"mmapv1", "wiredTiger" ] } }
{ "_id" : ObjectId("5b8164c2685b132ec4739503"), "book" : "RobotFramework" }
> db.startup_log.find();
{ "_id" : "yongqing-PC-1535190331203", "hostname" : "yongqing-PC", "startTime" :
 ISODate("2018-08-25T09:45:31Z"), "startTimeLocal" : "Sat Aug 25 17:45:31.203",
"cmdLine" : { "storage" : { "dbPath" : "D:\\MongoDB\\Server\\3.2\\data" }, "syst
emLog" : { "destination" : "file", "path" : "D:\\MongoDB\\Server\\3.2\\logs\\log
.log" } }, "pid" : NumberLong(8964), "buildinfo" : { "version" : "3.2.4", "gitVe
rsion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30", "targetMinOS" : "Windows Vi
sta/Windows Server 2008", "modules" : [ ], "allocator" : "tcmalloc", "javascript
Engine" : "mozjs", "sysInfo" : "deprecated", "versionArray" : [ 3, 2, 4, 0 ], "o
penssl" : { "running" : "disabled", "compiled" : "disabled" }, "buildEnvironment
" : { "distmod" : "", "distarch" : "x86_64", "cc" : "cl: Microsoft (R) C/C++ Opt
imizing Compiler Version 18.00.31101 for x64", "ccflags" : "/nologo /EHsc /W3 /w
d4355 /wd4800 /wd4267 /wd4244 /wd4290 /wd4068 /wd4351 /we4013 /we4099 /we4930 /Z
7 /errorReport:none /MT /O2 /Oy- /Gw /Gy /Zc:inline", "cxx" : "cl: Microsoft (R)
 C/C++ Optimizing Compiler Version 18.00.31101 for x64", "cxxflags" : "/TP", "li
nkflags" : "/nologo /DEBUG /INCREMENTAL:NO /LARGEADDRESSAWARE /OPT:REF", "target
_arch" : "x86_64", "target_os" : "windows" }, "bits" : 64, "debug" : false, "max
BsonObjectSize" : 16777216, "storageEngines" : [ "devnull", "ephemeralForTest",
"mmapv1", "wiredTiger" ] } }
{ "_id" : ObjectId("5b8164c2685b132ec4739503"), "book" : "robotFramework" }
> _
```

图 2-2-23

2.2.6 Remove Mongodb Records

Remove Mongodb Records 关键字用来删除指定 Collection 中的数据记录，接收[dbName | dbCollName | recordJSON]三个参数。

【示例】我们重新创建一个 capped 属性为 false 的 Collection，因为之前的 Collection 的 capped 属性为 true，会导致数据记录无法被删除。在客户端创建一个 Collection 的命令为 db.createCollection("RobotFramework",{capped : false})，如图 2-2-24 所示，新的名叫 RobotFramework 的 Collection 创建完成。

```
管理员: C:\Windows\system32\cmd.exe - mongo 127.0.0.1:27017
> db.createCollection("RobotFramework",{capped : false})
{ "ok" : 1 }
> show collections
RobotFramework
startup_log
```

图 2-2-24

创建完成后，通过客户端的 db.Robot Framework.find()命令进行查询，如图 2-2-25 所示。然后我们使用 Remove Mongodb Records 关键字来删除{"book":"robotFramework"}这条记录，如图 2-2-26 所示。

```
> use local
switched to db local
> db.RobotFramework.find()
{ "_id" : ObjectId("5b8202ee685b130624a2bc4d"), "book" : "RobotFramework" }
>
```

图 2-2-25

Connect To MongoDB	127.0.0.1	27017	2
${Json}	Set Variable	{"book":"RobotFramework"}	
Remove Mongodb Records	local	RobotFramework	${Json}
Disconnect From Mongodb			

图 2-2-26

运行结果如图 2-2-27 所示。

```
Starting test: RobotFrameworkTest1.TestSute13.TestCase007
20180826 09:40:05.945 :  INFO :
| Connect To MondoDB | dbHost | dbPort | dbMaxPoolSize | dbNetworktimeout |
dbDocClass | dbTZAware |
| Connect To MondoDB | 127.0.0.1 | 27017 | 2 | None | <type 'dict'> | False |
20180826 09:40:05.947 :  INFO : ${Json} = {"book":"RobotFramework"}
20180826 09:40:05.970 :  INFO : | ${allResults} | Remove MongoDB Records | local
| RobotFramework | {u'book': u'RobotFramework'} |
20180826 09:40:05.972 :  INFO : | Disconnect From MongoDB |
Ending test:   RobotFrameworkTest1.TestSute13.TestCase007
```

图 2-2-27

执行完成后，在客户端执行 db.RobotFramework.find()进行查询，如图 2-2-28 所示，从查询的结果可以看到{"book":"robotFramework"}这条记录已经被删除。

```
管理员: C:\Windows\system32\cmd.exe - mongo  127.0.0.1:27017
> db.RobotFramework.find()
{ "_id" : ObjectId("5b8202ee685b130624a2bc4d"), "book" : "RobotFramework" }
> db.RobotFramework.find()
> db.RobotFramework.find()
>
```

图 2-2-28

2.2.7 MongoDBLibrary 库的其他关键字

表 2-2-1 中列出了 MongoDBLibrary 库中其他关键字的使用示例。

表 2-2-1　MongoDBLibrary 库中其他关键字

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Drop Mongodb Database</td><td>该关键字用来删除指定的 MongoDB 数据库，接收[dbDelName]一个参数，示例：<table><tr><td>Drop Mongodb Database</td><td>local</td></tr></table></td></tr>
<tr><td>Drop Mongodb Collection</td><td>该关键字用来删除指定的 Collection，接收[dbName | dbCollName]两个参数，示例：<table><tr><td>Drop Mongodb Collection</td><td>local</td><td>RobotFramework</td></tr></table></td></tr>
<tr><td>Get Mongodb Collection Count</td><td>该关键字用来获取指定 Collection 下的数据记录总数，接收[dbName | dbCollName]两个参数，示例：<table><tr><td>${counts}</td><td>Get Mongodb Collection Count</td><td>local</td><td>RobotFramework</td></tr><tr><td>log</td><td>${counts}</td><td></td><td></td></tr></table></td></tr>
<tr><td>Retrieve And Update One Mongodb Record</td><td>该关键字用来获取并且更新指定的数据记录，接收[dbName | dbCollName | queryJSON | updateJSON | returnBeforeDocument=False]五个参数，示例：<table><tr><td>${queryJson}</td><td>Set Variable</td><td>{"book":"RobotFramework"}</td><td></td><td></td></tr><tr><td>${newJson}</td><td>Set Variable</td><td>{"$set":{"book":"robotFramework"}}</td><td></td><td></td></tr><tr><td>Retrieve And Update One Mongodb Record</td><td>local</td><td>RobotFramework</td><td>${queryJson}</td><td>${newJson}</td></tr></table></td></tr>
<tr><td>Retrieve Mongodb Records With Desired Fields</td><td>该关键字用来从 Collection 中根据指定的字段查询出对应的满足要求的数据记录，接收[dbName | dbCollName | recordJSON | fields | return__id=True | returnDocuments=False]六个参数，示例：<table><tr><td>${result}</td><td>Retrieve Mongodb Records With Desired Fields</td><td>local</td><td>RobotFramework</td><td>{}</td><td>book</td></tr><tr><td>log</td><td>${result}</td><td></td><td></td><td></td><td></td></tr></table></td></tr>
<tr><td>Retrieve Some Mongodb Records</td><td>该关键字用来从 Collection 中查询出根据指定 JSON 匹配到的数据记录，接收[dbName | dbCollName | recordJSON | returnDocuments=False]四个参数，示例：<table><tr><td>${result}</td><td>Retrieve Some Mongodb Records</td><td>local</td><td>RobotFramework</td><td>{"book":"robot Framework"}</td></tr><tr><td>log</td><td>${result}</td><td></td><td></td><td></td></tr></table></td></tr>
</table>

第 3 章 HTTP接口自动化测试

HTTP 接口自动化测试是常见的一种自动化测试需求和需要。在 Robot Framework 中，RequestsLibrary、HttpLibrary.HTTP、REST 等库都可以用来做 HTTP 接口方面的自动化测试。

3.1 HttpLibrary.HTTP 库的使用

通过访问 GitHub 地址 https://github.com/peritus/robotframework-httplibrary/#readme 可以下载和安装 HttpLibrary.HTTP 库，如图 3-1-1 所示。

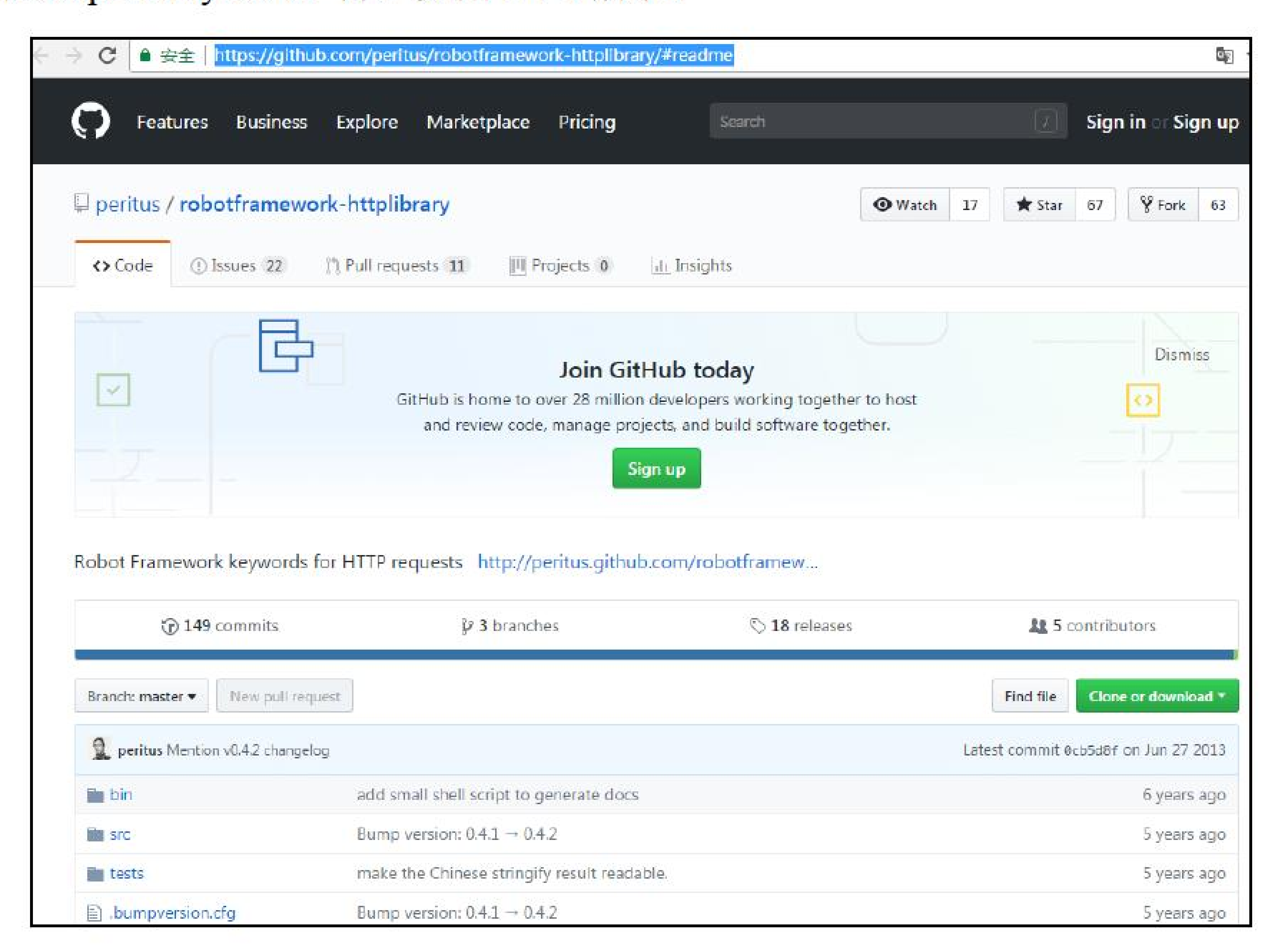


图 3-1-1

安装方式也可以通过 pip install --upgrade robotframework-httplibrary 来进行在线安装。安

装完成后，在使用时需要在 RIDE 中导入 HttpLibrary.HTTP 库，如图 3-1-2 所示。

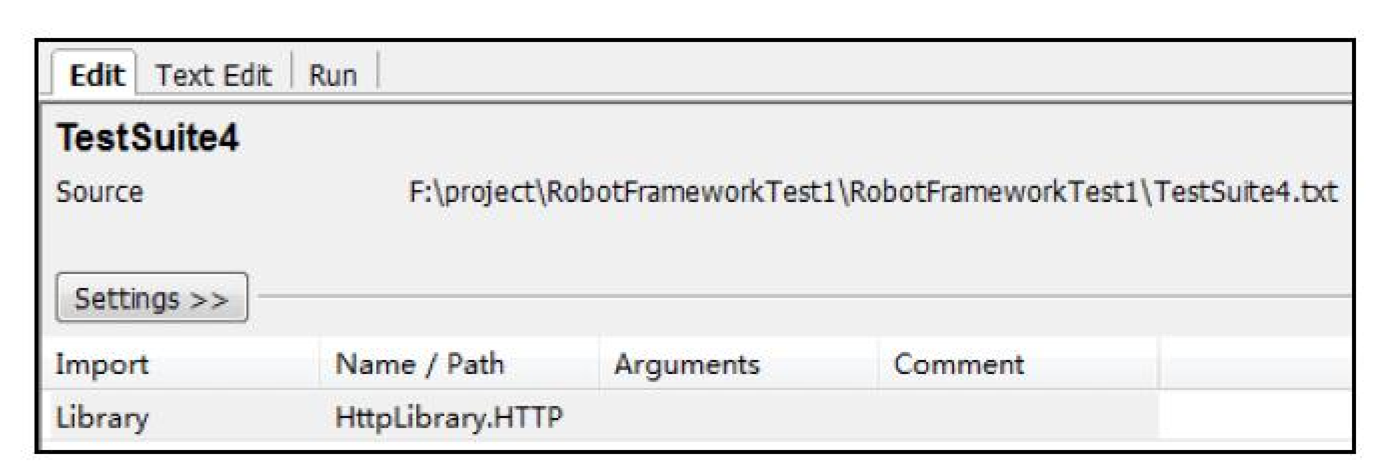


图 3-1-2

3.1.1　Create Http Context

要想使用 HttpLibrary，Create Http Context 关键字是必不可少的，其作用相当于创建了一个 HTTP 调用的环境。

Create Http Context 关键字需要接收两个参数：host 和 scheme。host 参数等同于 HTTP 协议中 Header（头部）中的 Host（指定请求的服务器的域名和端口号）。scheme 参数如果没有传入，就默认为 http，也可以传入 https（调用时使用 HTTPS 协议）。

【示例 1】scheme=http，并且 host 参数指定了域名和端口，如图 3-1-3 所示。



图 3-1-3

执行结果如图 3-1-4 所示。



图 3-1-4

【示例 2】scheme=https，并且 host 只指定了域名，没有指定端口号，直接使用 HTTPS 协议进行请求调用，如图 3-1-5 所示。



图 3-1-5

执行结果如图 3-1-6 所示。

```
- KEYWORD HttpLibrary.HTTP.Create Http Context host=www.baidu.com, scheme=https
  Documentation:        Sets the HTTP host to use for future requests. You must call this
  Start / End / Elapsed: 20170404 23:22:07.393 / 20170404 23:22:07.393 / 00:00:00.000
  23:22:07.393   INFO   Host for next HTTP request set to 'www.baidu.com'
  23:22:07.393   INFO   Scheme for next HTTP request set to 'https'
```

图 3-1-6

3.1.2 Get

在使用 Create Http Context 关键字创建了 HTTP 的调用环境后，就可以使用 Get 关键字来发送一个 HTTP 协议中常用的 get 请求了。

Get 关键字只接收一个 url 参数。该关键字的 API 中对 url 的原始描述是：`url` is the URL relative to the server root。

【示例 1】使用 get 请求访问百度主页，如图 3-1-7 所示。

```
Create Http Context    host=www.baidu.com:80   scheme=http
GET   /
```

Create Http Context	host=www.baidu.com:80	scheme=http
GET	/	

图 3-1-7

执行结果如图 3-1-8 所示。

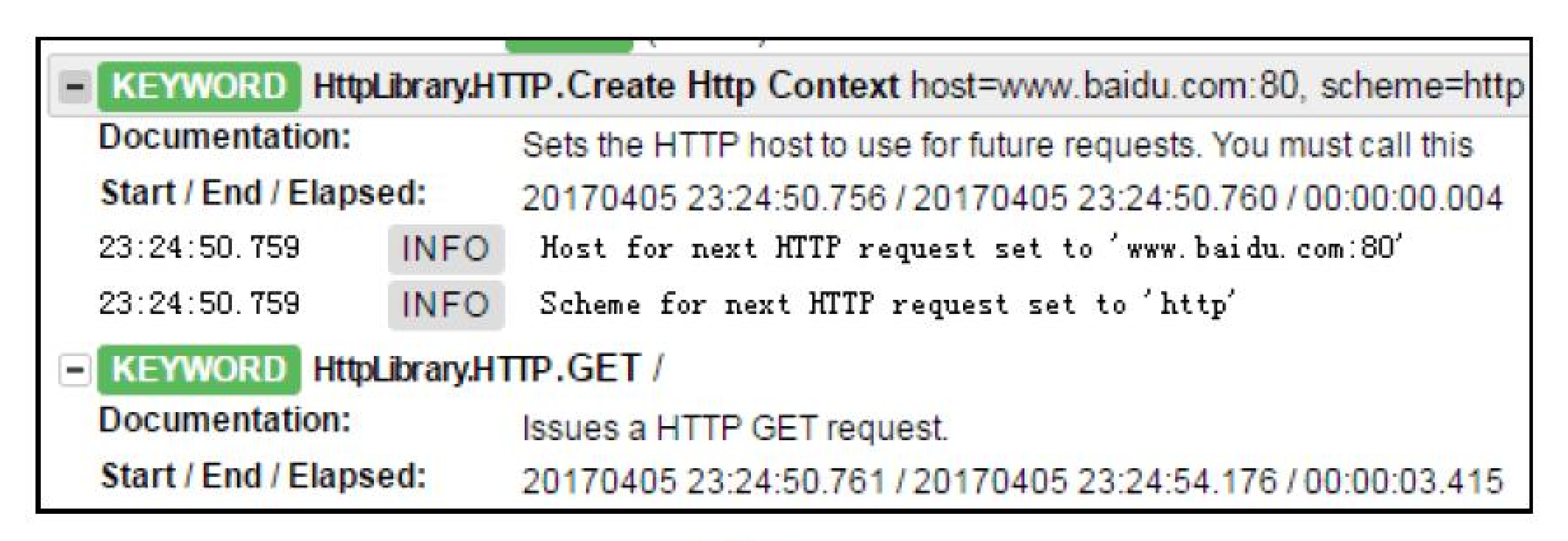


图 3-1-8

【示例 2】使用 get 请求访问 Robot Framework 主页下的 examples，如图 3-1-9 所示。

```
Create Http Context    robotframework.org  scheme=http
GET      /#examples
```

Create Http Context	robotframework.org	scheme=http
GET	/#examples	

图 3-1-9

执行结果如图 3-1-10 所示。

```
KEYWORD HttpLibrary.HTTP.Create Http Context robotframework.org, scheme=http
Documentation:       Sets the HTTP host to use for future requests. You must call this
Start / End / Elapsed:  20170405 23:38:15.781 / 20170405 23:38:15.781 / 00:00:00.000
23:38:15.781   INFO  Host for next HTTP request set to 'robotframework.org'
23:38:15.781   INFO  Scheme for next HTTP request set to 'http'
KEYWORD HttpLibrary.HTTP.GET /#examples
Documentation:       Issues a HTTP GET request.
Start / End / Elapsed:  20170405 23:38:15.781 / 20170405 23:38:19.047 / 00:00:03.266
KEYWORD ${body} = HttpLibrary.HTTP.Get Response Body
Documentation:       Get the response body.
Start / End / Elapsed:  20170405 23:38:19.057 / 20170405 23:38:19.107 / 00:00:00.050
```

图 3-1-10

【示例 3】使用 get 请求访问一个带有传入参数的示例，如图 3-1-11 所示。

```
Create Http Context    sp0.baidu.com    scheme=https
GET/5a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=htt&json=1&p=3&sid=22584_1441_21089_22176_20928&req=2&csor=0&pwd=ht&cb=jQuery110201988529936624046_1492182668411&_=1492182668416
${body} Get Response Body
log  ${body}
```

Create Http Context	sp0.baidu.com	scheme=https		
GET	/5a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=htt&json=1&p=3&sid=22584_1441_21089_22176_20928&req=2&csor=0&pwd=ht&cb=jQuery110201988529936624046_1492182668411&_=1492182668416			
${body}	Get Response Body			
log	${body}			

图 3-1-11

执行结果如图 3-1-12 所示。



图 3-1-12

3.1.3 Get Response Body

Get Response Body 关键字在上面已经用到了，服务器端在完成处理发出 HTTP 请求后会给出对应的响应结果，这时 Get Response Body 关键字就可以用来获取响应结果中的主体内容的。一般在 get 请求或者 post 请求发出后使用该关键字。

【示例】访问苏宁易购网站上的 HTTP 推荐接口，使用 Get Response Body 关键字获取返回的内容，如图 3-1-13 所示。

```
Create Http Context    tuijian.suning.com  scheme=http
GET
    /recommend-portal/recommendv2/biz.jsonp?callback=showFinal&parameter=%E7%
AC%94%E8%AE%B0%E6%9C%AC&sceneIds=2-1&count=5&cityId=9173&price=&brandCode=
${body} Get Response Body
log ${body}
```

Create Http Context	tuijian.suning.com	scheme=http		
GET	/recommend-portal/recommendv2/biz.jsonp?callback=showFinal&parameter=%E7%AC%94%E8%AE%B0%E6%9C%AC&sceneIds=2-1&count=5&cityId=9173&price=&brandCode=			
${body}	Get Response Body			
log	${body}			

图 3-1-13

执行结果如图 3-1-14 所示。

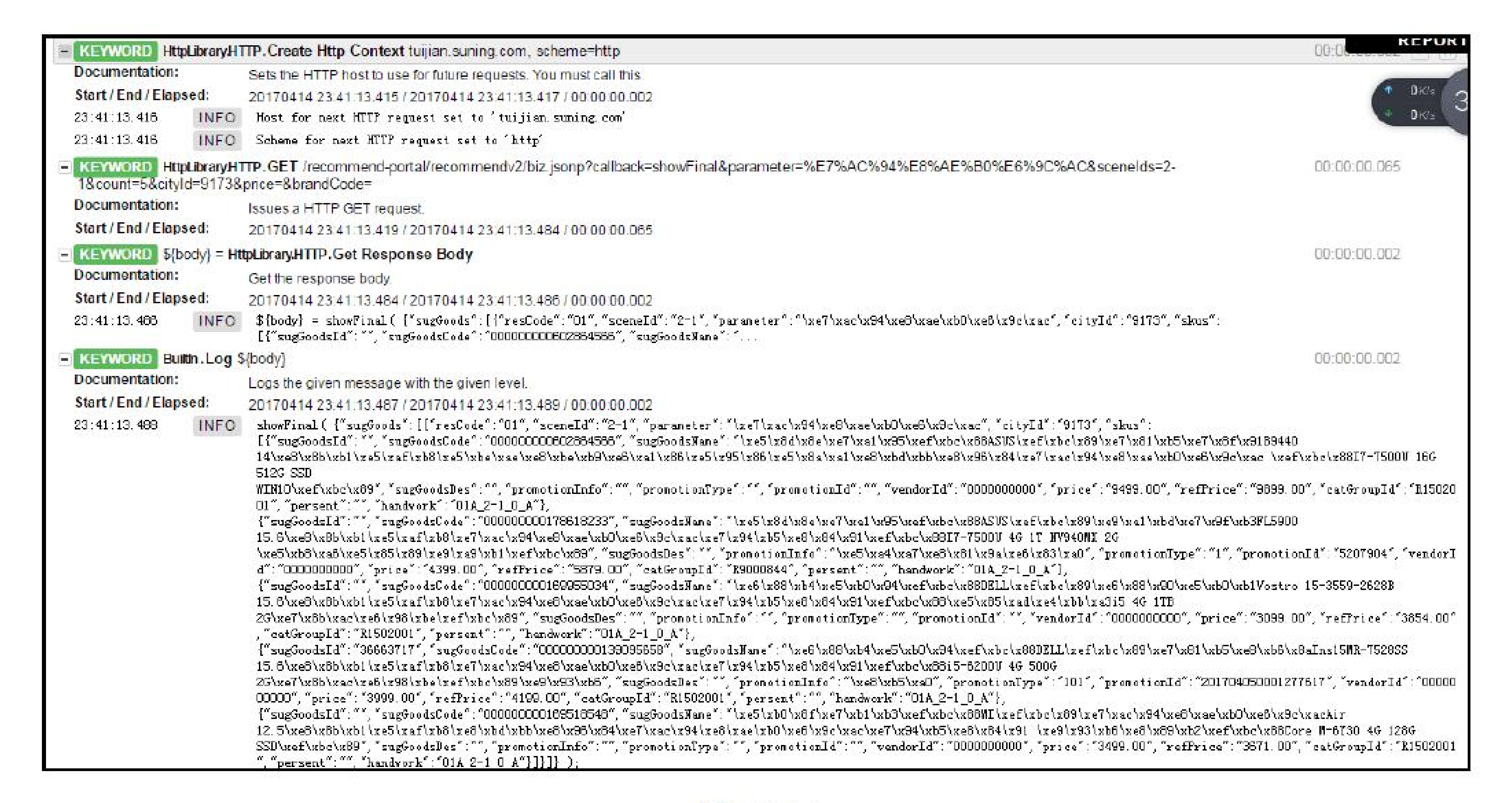


图 3-1-14

3.1.4 Get Response Status

Get Response Status 关键字用来获取 HTTP 请求返回的 HTTP 状态码。

【示例】访问苏宁易购网站上的 HTTP 推荐接口，使用 Get Response Status 关键字来获取

返回的 HTTP 状态码，如图 3-1-15 所示。

```
Create Http Context    tuijian.suning.com  scheme=http
GET
    /recommend-portal/recommendv2/biz.jsonp?callback=showFinal&parameter=%E7%
AC%94%E8%AE%B0%E6%9C%AC&sceneIds=2-1&count=5&cityId=9173&price=&brandCode=
${body} Get Response Status
log  ${body}
```

Create Http Context	tuijian.suning.com	scheme=http
GET	/recommend-portal/recommendv2/biz.jsonp?callback=showFinal&parameter=%E7%AC%94%E8%AE%B0%E6%9C%AC&sceneIds=2-1&count=5&cityId=9173&price=&brandCode=	
${body}	Get Response Status	
log	${body}	

图 3-1-15

执行结果如图 3-1-16 所示。

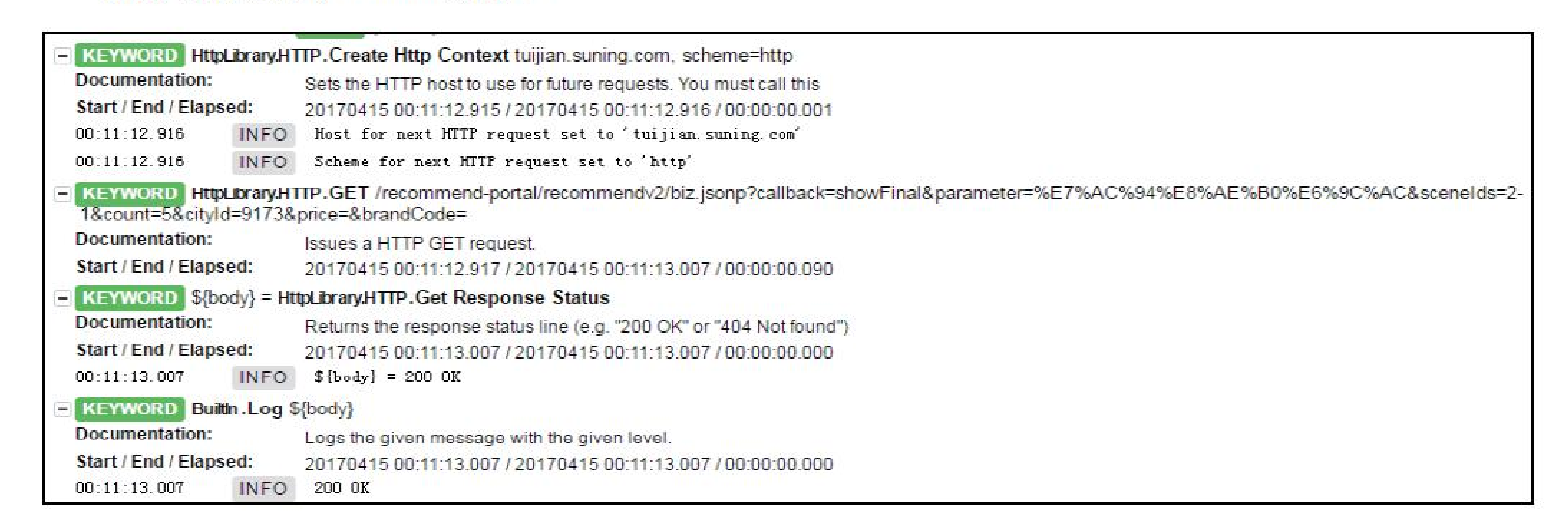


图 3-1-16

从获取到的结果看，我们获取到的状态码为 200 OK。

3.1.5　Get Response Header

Get Response Header 关键字用来获取 HTTP 请求返回的 HTTP 响应头部数据。常见的 Response Header 如表 3-1-1 所示。

表 3-1-1　常见的 Response Header

Header	解释	示例
Accept-Ranges	该字段表示服务器是否支持指定范围内的请求及哪种类型的分段请求	Accept-Ranges: bytes
Age	由原始服务器到代理缓存形成过程中的估算时间（以秒来计算，非负数）	Age: 12
Allow	对某种网络资源允许的有效的请求行为类型，若不允许则返回 http 状态码为 405	Allow: GET，HEAD，POST

（续表）

Header	解释	示例
Cache-Control	标明所有的缓存机制是否可以缓存及是哪种类型的缓存	Cache-Control: no-cache
Content-Encoding	服务器支持的返回内容的压缩编码类型	Content-Encoding: gzip
Content-Language	响应体的语言类型	Content-Language: en,zh
Content-Length	响应体的长度	Content-Length: 500
Content-Location	请求资源可替代的备用的其他请求地址	Content-Location: /home.htm
Content-MD5	返回资源的 MD5 加密校验值	Content-MD5: H3hlY1sgSW40RWcypXR9IK==
Content-Range	用于响应头，指定整个实体中一部分的插入位置。它也指示了整个实体的长度，在服务器向客户返回一个部分响应。它必须描述响应覆盖的范围和整个实体的长度，一般格式为： Content-Range: bytes (unit first byte pos) - [last byte pos]/[entity legth]	Content-Range: bytes 21010-47021/47022
Content-Type	返回内容的 MIME 类型	Content-Type:text/html;charset=utf-8
Date	原始服务器消息发出的时间	Date: Tue, 19 Nov 2014 20:12:31 GMT
ETag	请求变量的实体标签的当前值	ETag: "737060cd8c284d8af7ad308 2f209582d"
Expires	响应过期的日期和时间	Expires: Thu, 03 Dec 2014 19:00:00 GMT
Last-Modified	请求资源的最后修改时间	Last-Modified: Tue, 17 Nov 2015 13:25:16 GMT
Location	用来重定向接收方到非请求 URL 的位置来完成请求或标识新的资源，通俗地说就是 Web 服务器告诉浏览器。试图访问的对象已经被移到别的位置，到该头部指定的位置去取	Location: http://www.cnblogs.com/ laoqing/p/8542487.html
Pragma	一个在 HTTP/1.0 中规定的通用首部。Pragma: no-cache 兼容 HTTP 1.0，包括实现特定的指令，可应用到响应链上的任何接收方	Pragma: no-cache
Proxy-Authenticate	该参数标志认证方案和可应用到代理的该 URL 上的参数	Proxy-Authenticate: Basic

（续表）

Header	解释	示例
refresh	应用于重定向或一个新的资源被创造，在 10 秒之后重定向（由网景提出，被大部分浏览器支持）	Refresh: 10; url=http://www.cnblogs.com/laoqing/p/8542487.html
Retry-After	若实体暂时不可取，则通知客户端在指定时间之后再次尝试	Retry-After: 150
Server	Web 服务器软件名称	Server: Apache/1.2.30 (Unix) (Red-Hat/Linux)
Set-Cookie	用来设置 HTTP Cookie	Set-Cookie: UserID=Robot; Max-Age=3000; Version=1
Trailer	是一个响应首部，标志着允许发送方在分块发送的消息后面添加额外的元信息，这些添加的元信息可能是随着消息主体的发送动态生成的，例如消息的完整性校验、消息的数字签名认证或者消息经过处理之后的最终状态等	Trailer: Max-Forwards
Transfer-Encoding	文件传输编码类型	Transfer-Encoding:chunked
Vary	告诉下游代理是使用缓存响应还是从原始服务器请求	Vary: *
Via	告知代理客户该端响应是通过哪里发送的	Via: 1.0 fred, 1.1 nowhere.com (Apache/1.3)
Warning	是一个通用消息首部，包含当前消息状态可能存在的问题，在响应中可以出现多个 Warning 首部： ● 110:Response is Stale（由缓存服务器提供的响应已过期） ● 111:Revalidation Failed（由于无法访问服务器，响应验证失败） ● 112:Disconnected Operation（缓存服务器断开连接） ● 113:Heuristic Expiration（如果缓存服务器采用启发式方法，将缓存的有效时间设定为 24 小时，而在该响应的年龄超过 24 小时时发送） ● 199:Miscellaneous Warning（任意的、未明确指定的警告信息） ● 214:Transformation Applied（由代理服务器添加，如果它对返回的展现内容进行了任何转换，比如改变了内容编码、媒体类型等） ● 299:Miscellaneous Warning（与 199 类似，只不过指代的是持久化警告）	Warning: 111 Revalidation Failed

（续表）

Header	解释	示例
WWW-Authenticate	响应头定义了使用何种验证方式去获取对资源的连接	WWW-Authenticate: Basic

虽然 HTTP 的 Response Header 类型众多，但是并不是所有的请求都会返回表 3-1-1 中提到的每一种 Response Header 类型。

【示例】访问苏宁易购网站上的 HTTP 推荐接口，使用 Get Response Header 关键字获取返回的 HTTP 头部为 Content-Type 的数据，如图 3-1-17 所示。

```
Create Http Context    tuijian.suning.com  scheme=http
GET
    /recommend-portal/recommendv2/biz.jsonp?callback=showFinal&parameter=%E7%
AC%94%E8%AE%B0%E6%9C%AC&sceneIds=2-1&count=5&cityId=9173&price=&brandCode=
${header}    Get Response Header    Content-Type
log  ${header}
```

Create Http Context	tuijian.suning.com	scheme=http		
GET	/recommend-portal/recommendv2/biz.jsonp?callback=showFinal&parameter=%E7%AC%94%E8%AE%B0%E6%9C%AC&sceneIds=2-1&count=5&cityId=9173&price=&brandCode=			
${header}	Get Response Header	Content-Type		
log	${header}			

图 3-1-17

执行结果如图 3-1-18 所示。

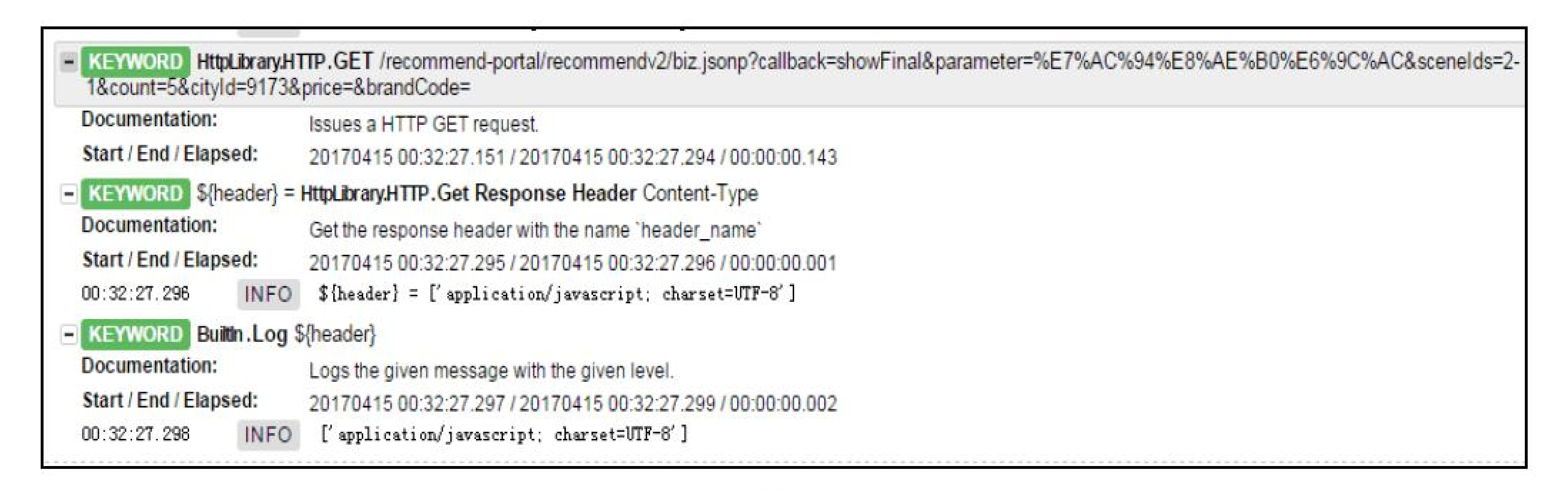


图 3-1-18

从返回的结果看，我们获取到的 Content-Type 为 application/javascript; charset=UTF-8。

3.1.6 Set Request Header

Set Request Header 关键字用来设置 HTTP 请求时的请求头部信息，接收[header_name | header_value] 两个参数。

【示例】设置 HTTP 请求时的 Referer（Referer 是 header 的一部分，当浏览器向 Web 服务器发送请求的时候，一般会带上 Referer，告诉服务器我是从哪个页面链接过来的，服务器

基于此可以获得一些处理信息。Referer 可以用来做一些安全方面的校验等）为 url：https://passport.cnblogs.com/user/signin?ReturnUrl=http://i.cnblogs.com（如图 3-1-19 所示）。

```
Set Request Header    Referer
    https://passport.cnblogs.com/user/signin?ReturnUrl=http://i.cnblogs.com/
```

Set Request Header	Referer	https://passport.cnblogs.com/user/signin?ReturnUrl=http://i.cnblogs.com/

图 3-1-19

执行结果如图 3-1-20 所示。

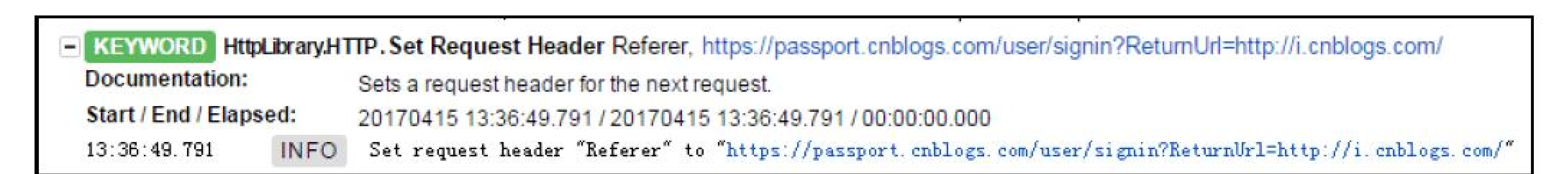


图 3-1-20

3.1.7　Set Request Body

Set Request Body 关键字用来设置 HTTP 请求时的 body 信息，尤其是在 post 请求时经常需要用到这个关键字。

该关键字接收[body]一个参数。

【示例】登录博客园（http://www.cnblogs.com/）时，设置登录请求时的 body 为：

```
{"input1":"V+bOQYKu0ZQXtauwxpGZC4YvvxwLI6/zA9U6Hx2JxXDZIpDXT2Fzravj7AiR5YDsSq
EUkYZFnB+komrnfzF5dzBHXN3FOxHdQ5oRoMdDHh/zdxTUuqg/9ev4V4yyT9T8V9fElmxs+BVrF2k
NZU35VtkaftQW8qn08Tl0GgdzEZY=","input2":"GAK4VTm2i+a/6bLHRIu8/oEeKJKav3SrU/DS
51300BmD/Xk6PEd0vk8GuLs6/obV3dl+y8Hub2Ey9+pO6ip53KIsNKPOeUly0P0aCIxMGK0jWcs+y
dClgeN0KRvxNlO/LTcW1hrXVcRBRAAhRcezdT2OTbYE4frZKIMShHqz+sE=","remember":false
}

Set Request Body
    {"input1":"V+bOQYKu0ZQXtauwxpGZC4YvvxwLI6/zA9U6Hx2JxXDZIpDXT2Fzravj7AiR5Y
DsSqEUkYZFnB+komrnfzF5dzBHXN3FOxHdQ5oRoMdDHh/zdxTUuqg/9ev4V4yyT9T8V9fElmxs+BV
rF2kNZU35VtkaftQW8qn08Tl0GgdzEZY=","input2":"GAK4VTm2i+a/6bLHRIu8/oEeKJKav3Sr
U/DS51300BmD/Xk6PEd0vk8GuLs6/obV3dl+y8Hub2Ey9+pO6ip53KIsNKPOeUly0P0aCIxMGK0jW
cs+ydClgeN0KRvxNlO/LTcW1hrXVcRBRAAhRcezdT2OTbYE4frZKIMShHqz+sE=","remember":f
alse}
```

执行结果如图 3-1-21 所示。

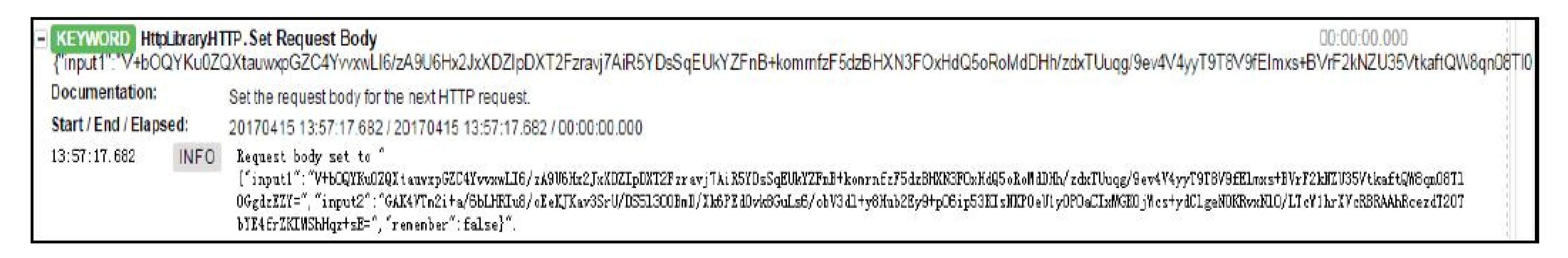


图 3-1-21

3.1.8 Post

在 HTTP 协议中，除了 get 请求外，另一个常用的就是 post 请求了。和 get 请求类似，post 请求接收[url]一个参数。

【示例】调用博客园（http://www.cnblogs.com/）的认证接口（https://passport.cnblogs.com/user/signin）进行用户登录认证，如图 3-1-22 所示。

```
Create Http Context    passport.cnblogs.com    scheme=https
Set Request Header    Cookie
    .Cnblogs.AspNetCore.Cookies=CfDJ8Mmb5OBERd5FqtiQlKZZIG41TLord2gXc8xTMoVr_
fYAteG89cxtvnObw-OyydeaaiQE8oRPEHPrSvWU32AGKXmVCEtOoQiuKIniNKqCvx2XNfChBRcA47
BDeEP4Il6EgeJ6ofQcdG62gN1c-xbk9bgcs7V1yYqUvNYW_tk2dd6Ffei77JuquWXwguCeGtVo4qt
GpUXLcP1YEWibXawE4ywbdovJTTihZD7yQB3OljzQjiUv2Q5BvcsqdzyKUxRcgxxSzcHqyDGGslL4
Sdvn7ho047ypsdFkgrafsfmzIpQ1;_ga=GA1.2.751066332.1492229820;SERVERID=9b2e527d
e1fc6430919cfb3051ec3e6c|1492230251|1492230186
Set Request Header    Referer
    https://passport.cnblogs.com/user/signin?ReturnUrl=http://i.cnblogs.com/
Set Request Body
    {"input1":"V+bOQYKu0ZQXtauwxpGZC4YvvxwLI6/zA9U6Hx2JxXDZIpDXT2Fzravj7AiR5Y
DsSqEUkYZFnB+komrnfzF5dzBHXN3FOxHdQ5oRoMdDHh/zdxTUuqg/9ev4V4yyT9T8V9fElmxs+BV
rF2kNZU35VtkaftQW8qn08Tl0GgdzEZY=","input2":"GAK4VTm2i+a/6bLHRIu8/oEeKJKav3Sr
U/DS513O0BmD/Xk6PEd0vk8GuLs6/obV3dl+y8Hub2Ey9+pO6ip53KIsNKPOeU1y0P0aCIxMGK0jW
cs+ydClgeN0KRvxNlO/LTcW1hrXVcRBRAAhRcezdT2OTbYE4frZKIMShHqz+sE=","remember":f
alse}
POST    /user/signin
${status}    Get Response status
${body} Get Response Body
log  ${body}
log  ${status}
```

Create Http Context	passport.cnblogs.com	scheme=https	
Set Request Header	Cookie	.Cnblogs.AspNetCore.Cookies=Cf	
Set Request Header	Referer	https://passport.cnblogs.com/user/signin?ReturnUrl=http://i.cnblogs.com/	
Set Request Body	{"input1":"V+bOQYKu0ZQXtauwxp		
POST	/user/signin		
${status}	Get Response status		
${body}	Get Response Body		
log	${body}		
log	${status}		

图 3-1-22

执行结果如图 3-1-23 所示。

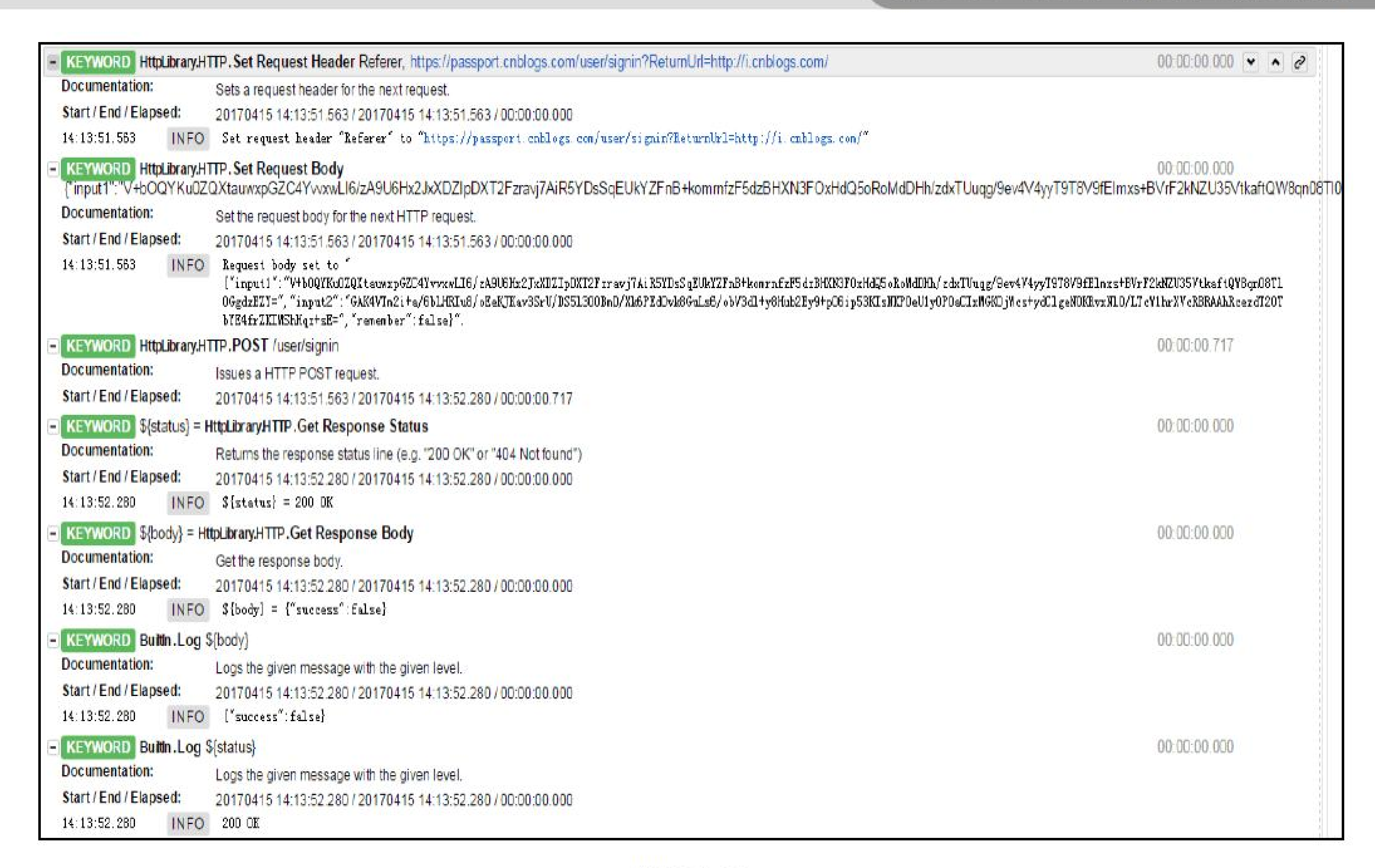


图 3-1-23

从执行结果看，我们认证失败了，请求返回的内容为{"success":false}，但是请求返回的HTTP code 为 200 OK，说明 HTTP post 请求已经发送成功了。

3.1.9　Follow Response

Follow Response 关键字用于处理 HTTP 中的重定向请求。常见的 HTTP 重定向请求包含 HTTP code 为 301 和 302 的两种重定向请求，如表 3-1-2 所示，代表着某个 URL 地址发生了转移。

表 3-1-2　301 和 302 两种重定向请求

HTTP code	说明
301	redirect: 301 代表永久性转移（Permanently Moved）
302	redirect: 302 代表暂时性转移（Temporarily Moved）

【示例】调用博客园（http://www.cnblogs.com/）的认证接口（https://passport.cnblogs.com/user/signin），进行用户登录认证。在请求时，没有设置 Cookie 这个 HTTP Header 时，HTTP code 会返回 302 ，在使用 Follow Response 关键字后会继续使用重定向后的 url 进行请求，如图 3-1-24 所示。

Create Http Context	passport.cnblogs.com	scheme=https	
Set Request Header Referer	https://passport.cnblogs.com/user/signin?ReturnUrl=http://i.cnblogs.com/		
Set Request Body	{"input1":"V+bOQYKu0ZQXtauwxpGZC4YvvxwLI6/zA9U6Hx2JxXDZIpDXT2Fzravj7AiR5YDsSqEUkYZFnB+komrnfzF5dzBHXN3FOxHdQ5oRoMdDHh/zdxTUuqg/9ev4V4yyT9T8V9fElmxs+BVrF2kNZU35VtkaftQW8qn08T10GgdzEZY=","input2":"GAK4VTm2i+a/6bLHRIu8/oEeKJKav3SrU/DS51300BmD/Xk6PEd0vk8GuLs6/obV3dl+y8Hub2Ey9+p06ip53KIsMKPOeU1y0P0aCIxMGK0jWcs+ydClgeN0KRvxN10/LTcW1hrXVcRBRAAhRcezdT20TbYE4frZKIMShHqz+sE=","remember":false}		
Follow Response			
${status}	**Get Response status**		
${body}	**Get Response Body**		
log	${body}		
log	${status}		

图 3-1-24

执行结果如图 3-1-25 所示。



图 3-1-25

从返回的结果看，在使用了 Follow Response 关键字后会继续使用重定向后的 url 进行请求，最后返回的 HTTP code 为 200 OK。

3.1.10 HttpLibrary.HTTP 库的其他关键字

表 3-1-3 中描述了 HTTPLibrary.HTTP 库中其他关键字的用法。

表 3-1-3　HTTPLibrary.HTTP 库中其他关键字的用法

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>DELETE</td><td>向服务器端发送 HTTP delete 请求，接收一个参数[url]，请求的方式和 post 请求非常类似，示例：<table><tr><td>DELETE</td><td>/ utils/config.htm</td></tr></table></td></tr>
<tr><td>HEAD</td><td>向服务器端发送 HTTP HEAD 请求，接收一个参数[url]，请求的方式和 get 请求非常类似，示例：<table><tr><td>HEAD</td><td>/ utils/config.htm</td></tr></table></td></tr>
<tr><td>PUT</td><td>向服务器端发送 HTTP PUT 请求，接收一个参数[url]，请求的方式和 post 请求非常类似，示例：<table><tr><td>PUT</td><td>/ _utils/config.htm</td></tr></table></td></tr>
<tr><td>Get Json Value</td><td>获取 Json 字符串中某个节点的值，示例：<table><tr><td>${value}</td><td>Get Json Value</td><td>{"foo": {"bar": [1,2,3]}}</td><td>/foo/bar</td></tr><tr><td>Should Be Equal</td><td>${value}</td><td>[1, 2, 3]</td><td></td></tr></table></td></tr>
<tr><td>Json Value Should Equal</td><td>一个断言关键字，用来判断 Json 字符串某个节点对应的值是不是和预期一致，示例：<table><tr><td>${json}</td><td>Set Variable</td><td>{"foo": {"bar": [1,2,3]}}</td><td></td></tr><tr><td>Json Value Should Equal</td><td>${json}</td><td>/foo/bar</td><td>[1, 2, 3]</td></tr></table></td></tr>
<tr><td>Json Value Should Not Equal</td><td>一个断言关键字，用来判断 Json 字符串某个节点对应的值是不是和预期不一致，当不一致时，执行成功，否则执行失败，示例：<table><tr><td>${json}</td><td>Set Variable</td><td>{"foo": {"bar": [1,2,3]}}</td><td></td></tr><tr><td>Json Value Should Not Equal</td><td>${json}</td><td>/foo/bar</td><td>[1, 2, 3]</td></tr></table></td></tr>
<tr><td>Log Response Body</td><td>打印出 HTTP 请求执行后的 Response Body 内容。该关键字一般在 post 请求或者 get 请求发出后使用，接收一个参数[log_level=INFO]，在没有传入日志级别参数时，默认使用 info 级别来打印日志</td></tr>
<tr><td>Log Response Headers</td><td>打印出 HTTP 请求执行后的 Response Headers 内容。该关键字一般在 post 请求或者 get 请求发出后使用，接收一个参数[log_level=INFO]，在没有传入日志级别参数时，默认使用 info 级别来打印日志</td></tr>
<tr><td>Log Response Status</td><td>打印出 HTTP 请求执行后的 Response Status 状态码，该关键字一般在 post 请求或者 get 请求发出后使用，接收一个参数[log_level=INFO]，在没有传入日志级别参数时，默认使用 info 级别来打印日志</td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Response Body Should Contain</td><td>一个断言关键字，用来判断 HTTP 请求响应后的 Response Body 中应该需要包含的内容，接收一个参数[should_contain（包含的内容）]，示例：<table><tr><td>GET</td><td>/foo.xml</td></tr><tr><td>Response Body Should Contain</td><td>version="1.0"</td></tr><tr><td>Response Body Should Contain</td><td>encoding="UTF-8"</td></tr></table></td></tr>
<tr><td>Response Header Should Equal</td><td>一个断言关键字，用来判断 HTTP 请求响应后的 Response Header 是否和预期一致，接收 [header_name | expected] 两个参数，示例：<table><tr><td>Response Header Should Equal</td><td>Content-Type</td><td>text/html; charset=utf-8</td></tr></table></td></tr>
<tr><td>Response Header Should Not Equal</td><td>一个断言关键字，和 Response Header Should Equal 刚好相反，用来判断 HTTP 请求响应后的 Response Header 是否和预期不一致，在不一致时，执行成功，否则执行失败，接收 [header_name | not_expected] 两个参数，示例：<table><tr><td>Response Header Should Not Equal</td><td>Content-Type</td><td>text/html; charset=utf-8</td></tr></table></td></tr>
<tr><td>Response Should Have Header</td><td>一个断言关键字，用来判断 HTTP 请求响应后的 Response Header 中是否包含预期的 header 名称，接收一个参数[header_name]，示例：<table><tr><td>Response Should Have Header</td><td>Content-Type</td></tr></table></td></tr>
<tr><td>Response Should Not Have Header</td><td>一个断言关键字，用来判断 HTTP 请求响应后的 Response Header 中是否不包含预期的 header 名称，接收一个参数[header_name]，示例：<table><tr><td>Response Should Not Have Header</td><td>Content-Type</td></tr></table></td></tr>
<tr><td>Response Status Code Should Equal</td><td>一个断言关键字，用来判断 HTTP 请求响应后的 Response status code 是否和预期一致，接收一个参数[status_code]，示例：<table><tr><td>Response Status Code Should Equal</td><td>200</td></tr></table></td></tr>
<tr><td>Response Status Code Should Not Equal</td><td>一个断言关键字，用来判断 HTTP 请求响应后的 Response status code 是否和预期不一致，若不一致，则执行成功，否则执行失败。该关键字接收一个参数[status_code]，示例：<table><tr><td>Response Status Code Should Not Equal</td><td>200</td></tr></table></td></tr>
<tr><td>Set Http Host</td><td>该关键字用来设置 HTTP 请求时的 host 名称，现在一般很少使用，推荐使用 Create HTTP Context 关键字来替代该关键字</td></tr>
<tr><td>Set Basic Auth</td><td>该关键字用来设置 HTTP 请求时的 Basic Auth(Basic Auth 简单点说明就是每次请求 API 时都提供用户的 username 和 password)，该关键字接收[username | password]两个参数</td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Set Json Value</td><td>该关键字用来设置 Json 字符串中某个节点的值，示例：
<table>
<tr><td>${json}</td><td>Set Json Value</td><td>{"foo": {"bar": [1,2,3]}}</td><td>/foo</td><td>12</td></tr>
<tr><td>Should Be Equal</td><td>${json}</td><td>{"foo": 12}</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Log Json</td><td>该关键字用来打印 Json 字符串的内容，示例：
<table>
<tr><td>Log Json</td><td>{"foo": {"bar": [1,2,3]}}</td><td>INFO</td></tr>
</table></td></tr>
<tr><td>Stringify Json</td><td>该关键字用来将数据转换为 Json 形式的字符串，示例：
<table>
<tr><td>${data}</td><td>Create List</td><td>a b c</td></tr>
<tr><td>${json_string}</td><td>Stringify JSON</td><td>${data}</td></tr>
<tr><td>log</td><td>${json_string}</td><td></td></tr>
</table></td></tr>
<tr><td>Show Response Body In Browser</td><td>使用默认的浏览器来显示最后一次 HTTP 请求的 Response Body</td></tr>
<tr><td>Should Be Valid Json</td><td>该关键字是一个断言关键字，用来判断某个字符串是不是一个有效的 Json 字符串，示例：
<table>
<tr><td>Should Be Valid Json</td><td>["a b c"]</td></tr>
</table></td></tr>
<tr><td>Parse Json</td><td>该关键字用来解析一个 Json 字符串，示例：
<table>
<tr><td>${json}</td><td>Parse Json</td><td>{"foo": {"bar": [1,2,3]}}</td></tr>
<tr><td>log</td><td>${json}</td><td></td></tr>
</table></td></tr>
<tr><td>Next Request Should Succeed</td><td>存在多次请求时，该关键字用来判断下一次请求应该需要执行成功。使用该关键字时，一般至少需要存在两次请求。在 HTTP 请求中，很多请求都有一个特性，即幂等，比如 get 请求、head 请求等都是幂等的请求，即执行一次和执行多次都应该是成功的，返回的结果应该都是一致的。在执行第二次 HTTP 请求时，返回的 HTTP code>=400 时，会认为第二次请求执行失败了。示例：
<table>
<tr><td>GET</td><td>/user/signin/aa.htm</td></tr>
<tr><td>Next Request Should Succeed</td><td></td></tr>
<tr><td>GET</td><td>/user/signin/aa.htm</td></tr>
</table></td></tr>
<tr><td>Next Request Should Not Succeed</td><td>存在多次请求时，该关键字用来判断下一次请求不应该执行成功。使用该关键字时，一般至少需要存在两次请求。HTTP 请求中，也有很多请求是非幂等的，比如常用的 post 请求就是一个典型的非幂等请求，即执行一次和执行多次，执行的结果不一定都是一致的。在执行第二次 HTTP 请求时，返回的 HTTP code<400 时，会认为第二次请求执行成功了。示例：
<table>
<tr><td>POST</td><td>/user/signin</td></tr>
<tr><td>Next Request Should Not Succeed</td><td></td></tr>
<tr><td>POST</td><td>/user/signin</td></tr>
</table></td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Next Request Should Have Status Code</td><td>存在多次请求时，该关键字用来判断下一次 HTTP 请求返回的 code 值是否和预期一致，示例:<table><tr><td>POST</td><td>/user/signin</td></tr><tr><td>Next Request Should Have Status Code</td><td>302</td></tr><tr><td>POST</td><td>/user/signin</td></tr></table></td></tr>
<tr><td>Next Request May Not Succeed</td><td>该关键字和 Next Request Should Succeed 关键字类似。使用该关键字时，一般至少需要存在两次请求，在下一次请求返回的 HTTP code>=400 时，会认为下一次请求执行失败了。示例：<table><tr><td>POST</td><td>/user/signin</td></tr><tr><td>Next Request May Not Succeed</td><td></td></tr><tr><td>POST</td><td>/user/signin</td></tr></table></td></tr>
</table>

3.2 RequestsLibrary 库的使用

通过访问 GitHub 地址 https://github.com/bulkan/robotframework-requests/#readme 可以下载和安装 RequestsLibrary 库，如图 3-2-1 所示。

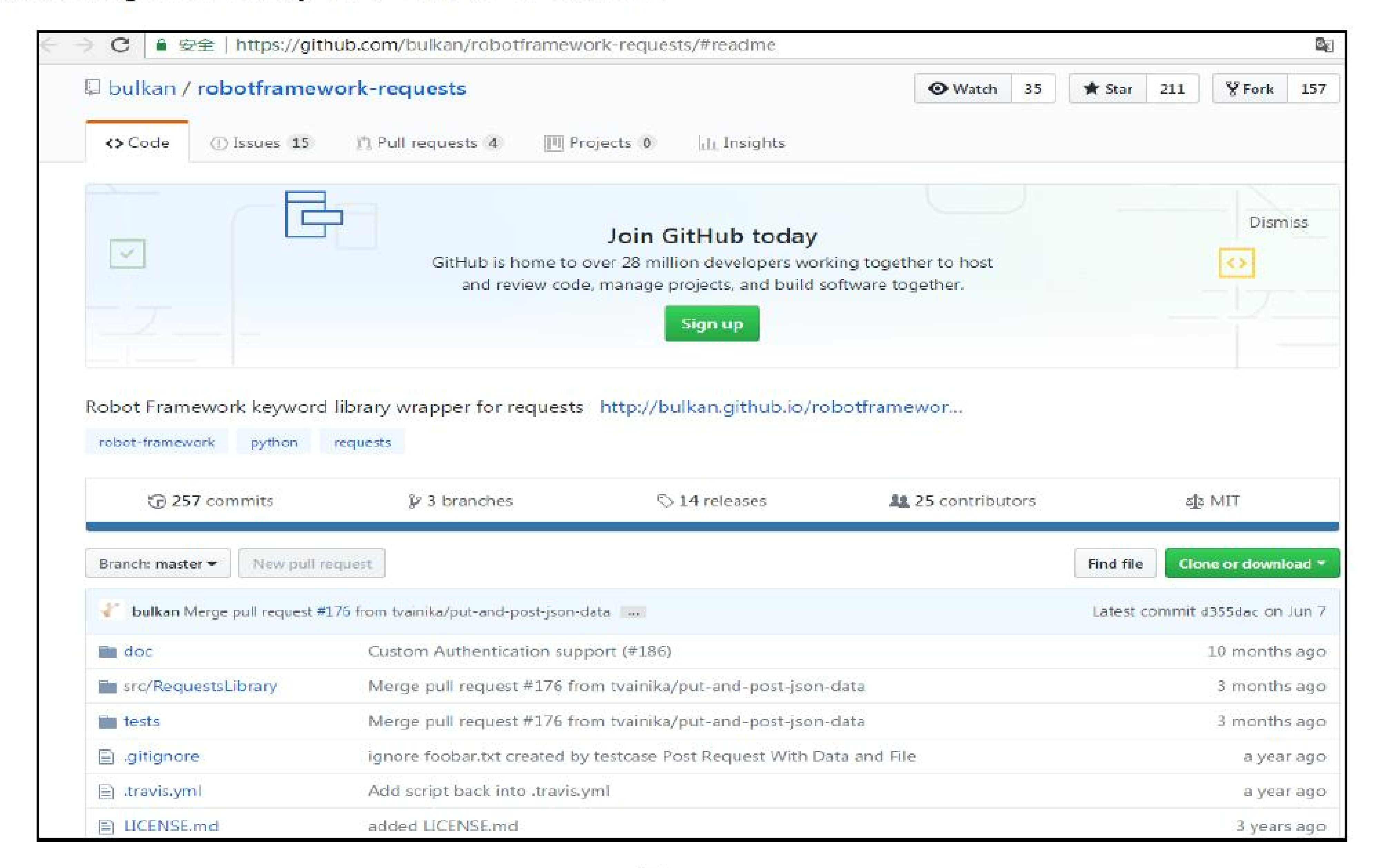


图 3-2-1

安装方式也可以通过如下命令行进行在线安装：

```
pip install -U requests
pip install -U robotframework-requests
```

安装完成后，使用时需要在 RIDE 中导入 RequestsLibrary 库，如图 3-2-2 所示。

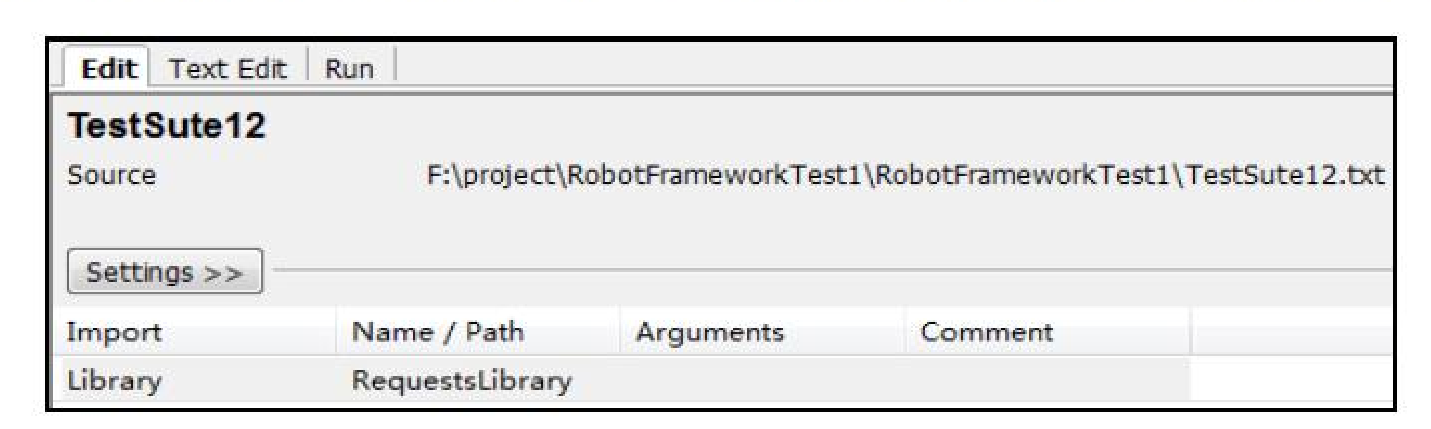


图 3-2-2

3.2.1　Create Session 和 Get Request

1. Create Session

Create Session 关键字用来向 HTTP 服务端创建一个 Session，接收[alias | url | headers={} | cookies=None | auth=None | timeout=None | proxies=None | verify=False]八个参数，相关介绍如表 3-2-1 所示。

表 3-2-1　Create Session 关键字的参数说明

参数名称	使用说明
url	HTTP 服务端的 url 地址
alias	RobotFramework 对创建的 Session 进行重命名
headers	HTTP 请求的 header，默认不传入时为空，若需要传入，则以字典的形式传入
auth	HTTP 服务基本 AUTH 验证时需要的用户名和密码，HTTP auth 是一种基础的用户验证
timeout	请求服务端时的超时时间设置
proxies	访问 HTTP 或者 HTTPS 服务时，需要的代理方式
cookies	需要设置的 cookies 值
verify	如果向服务端请求时需要验证证书，那么需要设置为 True，默认为 False

2. Get Request

Get Request 关键字用来在创建好 Session 的基础上向服务端发送一个 get 请求，接收[alias | uri | headers=None | params={} | allow_redirects=None] 五个参数，相关介绍如表 3-2-2 所示。

表 3-2-2　Get Request 关键字的参数说明

参数名称	使用说明
alias	Create Session 关键字执行后创建的 Session 别名，也就是在 Create Session 关键字执行时 alias 参数填入的值
uri	HTTP 服务端的 uri 地址
headers	HTTP 请求的 header，默认不传入时为空，若需要传入，则以字典的形式传入
params	HTTP 请求参数，若需要传入，则以字典的形式传入
allow_redirects	允许重定向的地址

【示例 1】使用 Create Session 和 Get Request 关键字向 http://robotframework.org/#libraries 发送一个 get 请求 ，如图 3-2-3 所示。

Create Session	RobotFramework	http://robotframework.org	
${resp}=	Get Request	RobotFramework	/#libraries
log	${resp}=		

图 3-2-3

运行结果如图 3-2-4 所示。

```
Starting test: RobotFrameworkTest1.TestSute12.TestCase0001
20180822 15:12:50.993 :  INFO : Creating Session using : alias=RobotFramework,
url=http://robotframework.org, headers={}, cookies=None, auth=None,
timeout=None, proxies=None, verify=False
20180822 15:12:51.071 :  INFO : Starting new HTTP connection (1):
robotframework.org
20180822 15:12:52.404 :  INFO : Get Request using : alias=RobotFramework,
uri=/#libraries, headers=None
20180822 15:12:52.405 :  INFO : ${resp} = <Response [200]>
20180822 15:12:52.406 :  INFO : <Response [200]>=
Ending test:   RobotFrameworkTest1.TestSute12.TestCase0001
```

图 3-2-4

【示例 2】我们使用 Create Session 和 Get Request 关键字来请求 https://tcc.taobao.com/cc/json/mobile_tel_segment.htm 网站提供的手机号码查询接口，这里的 headers 使用创建字典的形式传入，如图 3-2-5 所示。

${headers}	Create Dictionary	Accept=text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8			
${param}	Create Dictionary	tel=15150576251			
Create Session	tcc	https://tcc.taobao.com	${headers}		
${resp}=	Get Request	tcc	/cc/json/mobile_tel_segment.htm	${headers}	${param}
log	${resp}				

图 3-2-5

运行结果如图 3-2-6 所示。

```
Starting test: RobotFrameworkTest1.TestSute12.TestCase0002
20180822 17:22:01.781 :  INFO : ${headers} = {u'Accept':
u'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'
}
20180822 17:22:01.784 :  INFO : ${param} = {u'tel': u'15150576251'}
20180822 17:22:01.787 :  INFO : Creating Session using : alias=tcc,
url=https://tcc.taobao.com, headers={u'Accept':
u'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'
}, cookies=None, auth=None, timeout=None, proxies=None, verify=False
20180822 17:22:01.804 :  INFO : Starting new HTTPS connection (1): tcc.taobao.com
```

图 3-2-6

```
20180822 17:22:02.537 :  INFO : Get Request using : alias=tcc,
uri=/cc/json/mobile_tel_segment.htm, headers={u'Accept':
u'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'
}
20180822 17:22:02.538 :  INFO :
C:\Program Files
(x86)\python\lib\site-packages\requests\packages\urllib3\util\ssl_.py:315:
SNIMissingWarning: An HTTPS request has been made, but the SNI (Subject Name
Indication) extension to TLS is not available on this platform. This may cause
the server to present an incorrect TLS certificate, which can cause validation
failures. For more information, see
https://urllib3.readthedocs.org/en/latest/security.html#snimissingwarning.
  SNIMissingWarning
C:\Program Files
(x86)\python\lib\site-packages\requests\packages\urllib3\util\ssl_.py:120:
InsecurePlatformWarning: A true SSLContext object is not available. This prevents
urllib3 from configuring SSL appropriately and may cause certain SSL connections
to fail. For more information, see
https://urllib3.readthedocs.org/en/latest/security.html#insecureplatformwarni
ng.
  InsecurePlatformWarning
C:\Program Files
(x86)\python\lib\site-packages\requests\packages\urllib3\connectionpool.py:79
1: InsecureRequestWarning: Unverified HTTPS request is being made. Adding
certificate verification is strongly advised. See:
https://urllib3.readthedocs.org/en/latest/security.html
  InsecureRequestWarning)
20180822 17:22:02.539 :  INFO : ${resp} = <Response [200]>
20180822 17:22:02.540 :  INFO : <Response [200]>
Ending test:   RobotFrameworkTest1.TestSute12.TestCase0002
```

图 3-2-6（续）

3.2.2　Post Request

Post Request 关键字用来在创建好 Session 的基础上向服务端发送一个 post 请求，接收[alias | uri | data={} | headers=None | files={} | allow_redirects=None]六个参数。其中，data、headers、files 这几个参数，如果需要传入，那么都应该以字典的形式传入。

【示例】向 https://getman.cn 网站提供的测试接口提交一个 post 请求，如图 3-2-7 所示。

${data}	Create Dictionary	book=RobotFramework		
Create Session	getman	https://getman.cn		
${resp}=	Post Request	getman	/echo	${data}
log	${resp}			

图 3-2-7

执行结果如图 3-2-8 所示。

```
Starting test: RobotFrameworkTest1.TestSute12.TestCase0003
20180825 10:29:07.923 :  INFO : ${data} = {u'book': u'RobotFramework'}
20180825 10:29:07.925 :  INFO : Creating Session using : alias=getman,
url=https://getman.cn, headers={}, cookies=None, auth=None, timeout=None,
proxies=None, verify=False
20180825 10:29:07.941 :  INFO : Starting new HTTPS connection (1): getman.cn
```

图 3-2-8

```
20180825 10:29:08.656 :  INFO : Post Request using : alias=getman, uri=/echo,
data={u'book': u'RobotFramework'}, headers=None, file={}, allow s
_redirects=True
20180825 10:29:08.658 :  INFO : ${resp} = <Response [200]>
20180825 10:29:08.660 :  INFO : <Response [200]>
Ending test:   RobotFrameworkTest1.TestSute12.TestCase0003
```

图 3-2-8（续）

3.2.3　RequestsLibrary 库的其他关键字

表 3-2-3 中列出了 RequestsLibrary 库中其他关键字的使用方式。

表 3-2-3　RequestLibrary 库中其他关键字的使用方式

关键字	使用描述
Delete	模拟向服务端发送一个 Delete 请求，接收[alias | uri | data=() | headers=None | allow_redirects=None]五个参数，不推荐使用 Delete 关键字来发送 Delete 请求，推荐使用 Delete Request 关键字来发送 Delete 请求
Delete Request	模拟向服务端发送一个 Delete 请求，接收[alias | uri | data=() | headers=None | allow_redirects=None]五个参数，推荐使用该关键字来发送 Delete 请求
Get	模拟发送一个 Get 请求，接收[alias | uri | headers=None | params={} | allow_redirects=None]五个参数，推荐使用 Get Request 关键字来发送 Get 请求
Head	模拟向服务端发送一个 Head 请求，接收[alias | uri | headers=None | allow_redirects=None]四个参数，不推荐使用该关键字来发送 Head 请求，推荐使用 Head Request 关键字来发送 Head 请求
Head Request	模拟向服务端发送一个 Head 请求，接收[alias | uri | headers=None | allow_redirects=None]四个参数，推荐使用该关键字来发送 Head 请求
Options	模拟向服务端发送一个 Options 请求，接收[alias | uri | headers=None | allow_redirects=None]四个参数，不推荐使用该关键字来发送 Options 请求，推荐使用 Options Request 关键字来发送 Options 请求
Options Request	模拟向服务端发送一个 Options 请求，接收[alias | uri | headers=None | allow_redirects=None]四个参数，推荐使用该关键字来发送 Options 请求
Patch	模拟向服务端发送一个 Patch 请求，接收[alias | uri | data={} | headers=None | files={} | allow_redirects=None]六个参数，不推荐使用该关键字来发送 Patch 请求，推荐使用 Patch Request 关键字来发送 Patch 请求
Patch Request	模拟向服务端发送一个 Patch 请求，接收[alias | uri | data={} | headers=None | files={} | allow_redirects=None]六个参数，推荐使用该关键字来进行 Patch 请求发送
Put	模拟向服务端发送一个 Put 请求，接收 [alias | uri | data=None | headers=None | allow_redirects=None]五个参数，不推荐使用该关键字来发送 Put 请求，推荐使用 Put Request 关键字来发送 Put 请求
Put Request	模拟向服务端发送一个 put 请求，接收[alias | uri | data=None | headers=None | allow_redirects=None]五个参数，推荐使用该关键字来进行 Put 请求发送
To Json	将一个字符串转换为一个 Json 形式的对象，接收[content | pretty_print=False]两个参数
Delete All Sessions	删除创建的所有 Session 对象，和 Create Session 关键字是相对应的

3.3 RESTinstance 库的使用

RESTinstance 库主要提供了用于 Restful 服务的 JSON 请求报文方式的关键字，可以通过在 cmd 命令行中输入“pip install --upgrade RESTinstance”进行在线安装。安装完成后，通过如图 3-3-1 所示的方式导入 RESTLibrary。

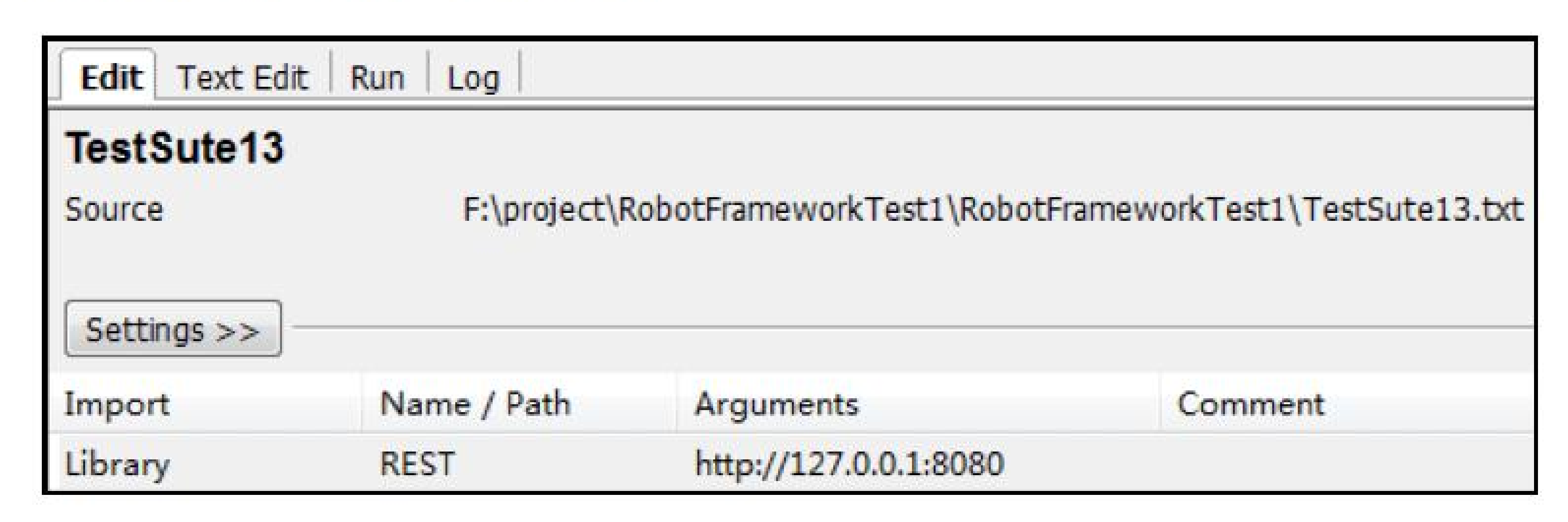


图 3-3-1

RESTLibrary 导入时，可以跟上需要请求的 RESTful 服务的地址。

RESTinstance 库提供的主要常用关键字如表 3-3-1 所示。

表 3-3-1　RESTinstance 库提供的主要常用关键字

<table>
<tr><th>关键字</th><th>使用说明</th></tr>
<tr><td>Post</td><td>以 RESTful 的形式向指定的服务器 uri 发送一个 post 请求，请求时 request body 需要是 JSON 的形式，示例：<table><tr><td>Post</td><td>/testRobot</td><td>{"book":"Robot"}</td></tr></table></td></tr>
<tr><td>Get</td><td>以 RESTful 的形式向指定的服务器 uri 发送一个 get 请求，示例：<table><tr><td>Get</td><td>/bookName</td></tr></table></td></tr>
<tr><td>Put</td><td>以 RESTful 的形式向指定的服务器 uri 发送一个 Put 请求，使用的方式和 Post 请求类似</td></tr>
<tr><td>Delete</td><td>以 RESTful 的形式向指定的服务器 uri 发送一个 Delete 请求，示例：<table><tr><td>Delete</td><td>/books/3</td></tr></table></td></tr>
<tr><td>Set Headers</td><td>该关键字用来设置 HTTP 请求时的请求 Headers，示例：<table><tr><td>Set Headers</td><td>{"content-type": "application/json"}</td></tr></table></td></tr>
<tr><td>Array</td><td>验证一个 RESTful 请求返回的响应结果是不是一个期望要求的数组，示例：<table><tr><td>Array</td><td>response body</td><td>maxItems=20</td></tr></table></td></tr>
<tr><td>Integer</td><td>用来断言指定的字段的返回值是否和预期结果一致，一般可以用来判断 RESTful 请求返回的 HTTP code，示例：<table><tr><td>Integer</td><td>response status</td><td>200</td></tr></table></td></tr>
</table>

第 4 章 移动手机自动化测试

4.1 Appium 介绍

Appium 是一个开源、跨平台的自动化测试工具，可以用来测试 Native 及混合的移动端应用，Appium 可以支持 IOS、Android 及 FirefoxOS 平台，Appium 为了实现自身提出的理念：不必局限于某种语言或者框架来写/运行测试脚本，以及一个移动自动化的框架不应该在接口上重复造轮子。它把 IOS、Android 等自身提供的第三方框架都封装成了一套 API，即 WebDriver。API.WebDriver（Selenium WebDriver，Appium 对此进行了扩展） 指定了客户端到服务端的协议，通过这种客户端/服务端的架构可以使用任何语言来编写客户端，向服务端发送恰当的 HTTP 请求，只要 client 能够发送 http 请求给 server，那么 client 用什么语言来实现都是可以的，这就是 appium 及 webdriver 如何做到支持多语言的。

Appium 的核心是一个 Web 服务器，提供了一套 REST 的接口。它收到客户端的连接，监听到命令，接着在移动设备上执行这些命令，然后将执行结果放在 HTTP 响应中返还给客户端。

Appium 的下载地址为 http://appium.io/downloads.html，从中可以下载最新的安装版本和相应的 Library 库，如图 4-1-1 所示。

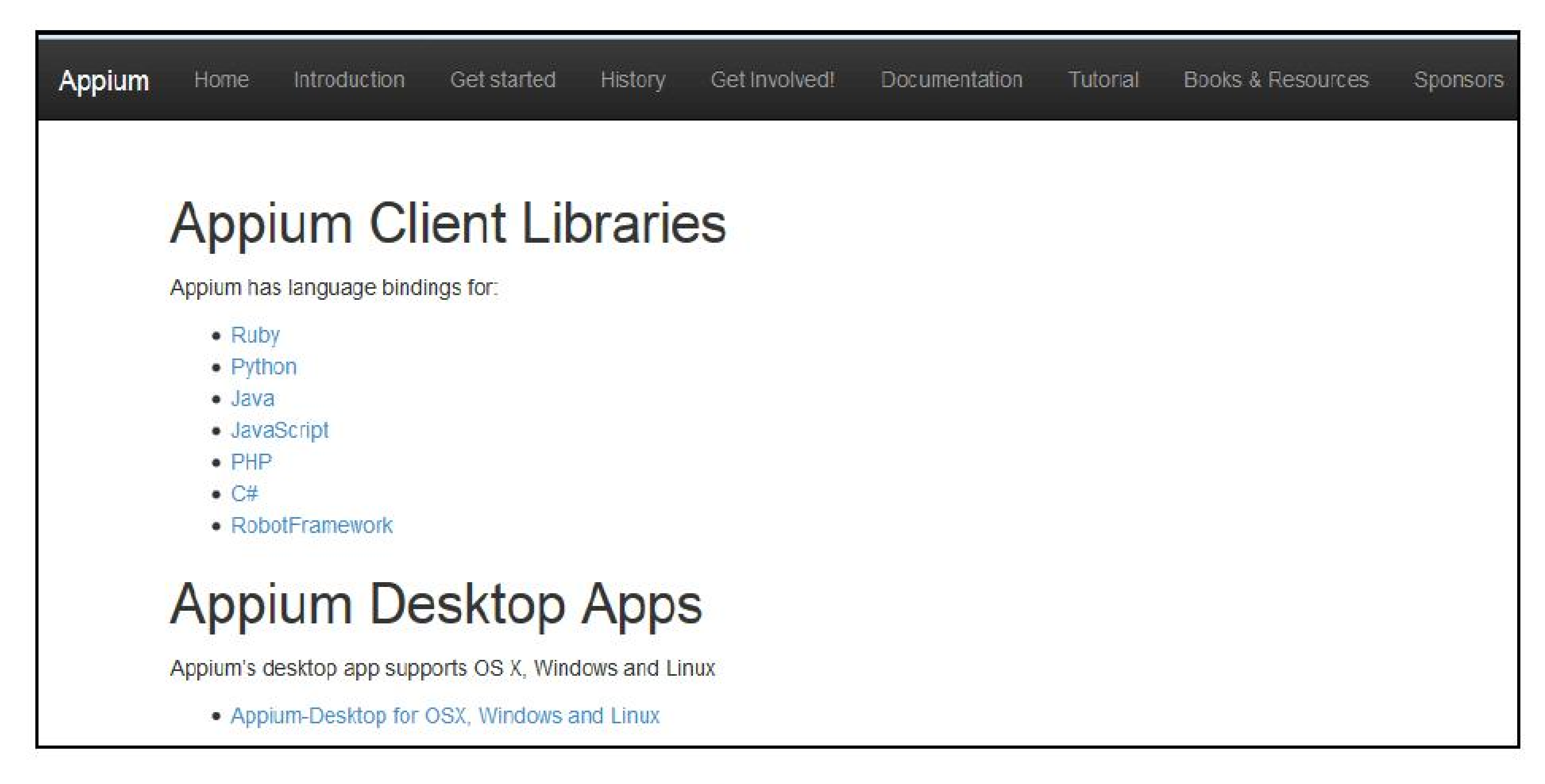


图 4-1-1

注：本章参考了 Appium 官网 http://appium.io/slate/cn/master/ 关于 Appium 的相关使用介绍。

Appium 的相关介绍文档可以通过访问 http://appium.io/slate/cn/master/获取到，如图 4-1-2 所示。



图 4-1-2

Appium 真正的工作引擎其实是第三方自动化框架：

- iOS：苹果的 UIAutomation。
- Android 4.2+：谷歌的 UiAutomator。
- Android 2.3+：谷歌的 Instrumentation。Instrumentation 由单独的项目 Selendroid 提供支持。

4.1.1 Appium 中的常用术语

Appium 中的常用术语有以下几种。

1. Session

Appium 自动化建立在一个 Session 上运行，客户端初始化一个 Seesion 来与服务端进行请求和交互，客户端会发送一个 POST 请求给服务端，请求中包含一个 JSON 对象，被称作"desired Capabilities"。然后，服务端就会开启一个自动化的 Session，并且返回一个 Session ID 给客户端，客户端的后续请求都会带上该 Session ID 给服务端做识别。

2. Desired Capabilities

Desired Capabilities 是一些键值对的集合（比如，一个 map 或者 hashmap），客户端将这些键值对发给服务端，告诉服务端需要怎么执行测试。比如，我们可以把 platformNamecapability 设置为 Android，告诉 Appium 服务端，我们想要一个 Android 的

Session，而不是一个 iOS 的。也可以设置 safariAllowPopups capability 为 True，确保在 Safari 自动化 Session 中可以使用 JavaScript 来打开新窗口。

3. Appium Server

Appium Server 是基于 Node.js 实现的，可以使用 NPM 直接安装。

4.1.2 Appium 服务关键字

Appium 服务关键字如下：

```
usage: appium.js [-h] [-v] [--shell]
              [--localizable-strings-dir LOCALIZABLESTRINGSDIR] [--app APP]
              [--ipa IPA] [-U UDID] [-a ADDRESS] [-p PORT]
              [-ca CALLBACKADDRESS] [-cp CALLBACKPORT] [-bp BOOTSTRAPPORT]
              [-k] [-r BACKENDRETRIES] [--session-override] [--full-reset]
              [--no-reset] [-l] [-lt LAUNCHTIMEOUT] [-g LOG]
              [--log-level {info,info:debug,info:info,info:warn,info:error,wa
rn,warn:debug,warn:info,warn:warn,warn:error,error,error:debug,error:info,err
or:
warn,error:error,debug,debug:debug,debug:info,debug:warn,debug:error}]
              [--log-timestamp] [--local-timezone] [--log-no-colors]
              [-G WEBHOOK] [--native-instruments-lib]
              [--app-pkg ANDROIDPACKAGE] [--app-activity ANDROIDACTIVITY]
              [--app-wait-package ANDROIDWAITPACKAGE]
              [--app-wait-activity ANDROIDWAITACTIVITY]
              [--android-coverage ANDROIDCOVERAGE] [--avd AVD]
              [--avd-args AVDARGS]
              [--device-ready-timeout ANDROIDDEVICEREADYTIMEOUT] [--safari]
              [--device-name DEVICENAME] [--platform-name PLATFORMNAME]
              [--platform-version PLATFORMVERSION]
              [--automation-name AUTOMATIONNAME]
              [--browser-name BROWSERNAME] [--default-device]
              [--force-iphone] [--force-ipad] [--language LANGUAGE]
              [--locale LOCALE] [--calendar-format CALENDARFORMAT]
              [--orientation ORIENTATION]
              [--tracetemplate AUTOMATIONTRACETEMPLATEPATH]
              [--instruments INSTRUMENTSPATH] [--show-sim-log]
              [--show-ios-log] [--nodeconfig NODECONFIG] [-ra ROBOTADDRESS]
              [-rp ROBOTPORT] [--selendroid-port SELENDROIDPORT]
              [--chromedriver-port CHROMEDRIVERPORT]
              [--chromedriver-executable CHROMEDRIVEREXECUTABLE]
              [--use-keystore] [--keystore-path KEYSTOREPATH]
              [--keystore-password KEYSTOREPASSWORD] [--key-alias KEYALIAS]
              [--key-password KEYPASSWORD] [--show-config]
              [--no-perms-check] [--command-timeout DEFAULTCOMMANDTIMEOUT]
              [--keep-keychains] [--strict-caps] [--isolate-sim-device]
              [--tmp TMPDIR] [--trace-dir TRACEDIR]
              [--intent-action INTENTACTION]
              [--intent-category INTENTCATEGORY]
```

```
               [--intent-flags INTENTFLAGS]
               [--intent-args OPTIONALINTENTARGUMENTS]
               [--dont-stop-app-on-reset] [--debug-log-spacing]
               [--suppress-adb-kill-server] [--async-trace]
A webdriver-compatible server for use with native and hybrid iOS and Android
applications.
Optional arguments:
  -h, --help            Show this help message and exit.
  -v, --version         Show program's version number and exit.
  --shell               Enter REPL mode
  --localizable-strings-dir LOCALIZABLESTRINGSDIR
                        IOS only: the relative path of the dir where
                        Localizable.strings file resides
  --app APP             IOS: abs path to simulator-compiled .app file or the
                        bundle_id of the desired target on device; Android:
                        abs path to .apk file
  --ipa IPA             (IOS-only) abs path to compiled .ipa file
  -U UDID, --udid UDID  Unique device identifier of the connected physical
                        device
  -a ADDRESS, --address ADDRESS
                        IP Address to listen on
  -p PORT, --port PORT  port to listen on
  -ca CALLBACKADDRESS, --callback-address CALLBACKADDRESS
                        callback IP Address (default: same as --address)
  -cp CALLBACKPORT, --callback-port CALLBACKPORT
                        callback port (default: same as port)
  -bp BOOTSTRAPPORT, --bootstrap-port BOOTSTRAPPORT
                        (Android-only) port to use on device to talk to Appium
  -k, --keep-artifacts  [DEPRECATED] no effect, trace is now in tmp dir by
                        default and is cleared before each run. Please also
                        refer to the --trace-dir flag.
  -r BACKENDRETRIES, --backend-retries BACKENDRETRIES
                        (iOS-only) How many times to retry launching
                        Instruments before saying it crashed or timed out
  --session-override    Enables session override (clobbering)
  --full-reset          (iOS) Delete the entire simulator folder. (Android)
                        Reset app state by uninstalling app instead of
                        clearing app data. On Android, this will also remove
                        the app after the session is complete.
  --no-reset            Don't reset app state between sessions (IOS: don't
                        delete app plist files; Android: don't uninstall app
                        before new session)
  -l, --pre-launch      Pre-launch the application before allowing the first
                        session (Requires --app and, for Android, --app-pkg
                        and --app-activity)
  -lt LAUNCHTIMEOUT, --launch-timeout LAUNCHTIMEOUT
                        (iOS-only) how long in ms to wait for Instruments to
                        launch
  -g LOG, --log LOG     Also send log output to this file
  --log-level
```

```
{info,info:debug,info:info,info:warn,info:error,warn,warn:debug,wa
rn:info,warn:warn,warn:error,error,error:debug,error:info,error:warn,error:er
ror
,debug,debug:debug,debug:info,debug:warn,debug:error}
                    log level; default (console[:file]): debug[:debug]
  --log-timestamp       Show timestamps in console output
  --local-timezone      Use local timezone for timestamps
  --log-no-colors       Don't use colors in console output
  -G WEBHOOK, --webhook WEBHOOK
                    Also send log output to this HTTP listener
  --native-instruments-lib
                    (IOS-only) IOS has a weird built-in unavoidable delay.
                     We patch this in appium. If you do not want it
                    patched, pass in this flag.
  --app-pkg ANDROIDPACKAGE
                    (Android-only) Java package of the Android app you
                    want to run (e.g., com.example.android.myApp)
  --app-activity ANDROIDACTIVITY
                    (Android-only) Activity name for the Android activity
                    you want to launch from your package (e.g.,
                    MainActivity)
  --app-wait-package ANDROIDWAITPACKAGE
                    (Android-only) Package name for the Android activity
                    you want to wait for (e.g., com.example.android.myApp)
  --app-wait-activity ANDROIDWAITACTIVITY
                    (Android-only) Activity name for the Android activity
                    you want to wait for (e.g., SplashActivity)
  --android-coverage ANDROIDCOVERAGE
                    (Android-only) Fully qualified instrumentation class.
                    Passed to -w in adb shell am instrument -e coverage
                    true -w
  --avd AVD           (Android-only) Name of the avd to launch
  --avd-args AVDARGS    (Android-only) Additional emulator arguments to
                    launch the avd
  --device-ready-timeout ANDROIDDEVICEREADYTIMEOUT
                    (Android-only) Timeout in seconds while waiting for
                    device to become ready
  --safari            (IOS-Only) Use the safari app
  --device-name DEVICENAME
                    Name of the mobile device to use
  --platform-name PLATFORMNAME
                    Name of the mobile platform: iOS, Android, or
                    FirefoxOS
  --platform-version PLATFORMVERSION
                    Version of the mobile platform
  --automation-name AUTOMATIONNAME
                    Name of the automation tool: Appium or Selendroid
  --browser-name BROWSERNAME
                    Name of the mobile browser: Safari or Chrome
  --default-device, -dd
```

```
                   (IOS-Simulator-only) use the default simulator that
                   instruments launches on its own
  --force-iphone       (IOS-only) Use the iPhone Simulator no matter what
                   the app wants
  --force-ipad         (IOS-only) Use the iPad Simulator no matter what the
                   app wants
  --language LANGUAGE   Language for the iOS simulator / Android Emulator
  --locale LOCALE       Locale for the iOS simulator / Android Emulator
  --calendar-format CALENDARFORMAT
                   (IOS-only) calendar format for the iOS simulator
  --orientation ORIENTATION
                   (IOS-only) use LANDSCAPE or PORTRAIT to initialize
                   all requests to this orientation
  --tracetemplate AUTOMATIONTRACETEMPLATEPATH
                   (IOS-only) .tracetemplate file to use with Instruments
  --instruments INSTRUMENTSPATH
                   (IOS-only) path to instruments binary
  --show-sim-log       (IOS-only) if set, the iOS simulator log will be
                   written to the console
  --show-ios-log       (IOS-only) if set, the iOS system log will be written
                   to the console
  --nodeconfig NODECONFIG
                   Configuration JSON file to register appium with
                   selenium grid
  -ra ROBOTADDRESS, --robot-address ROBOTADDRESS
                   IP Address of robot
  -rp ROBOTPORT, --robot-port ROBOTPORT
                   port for robot
  --selendroid-port SELENDROIDPORT
                   Local port used for communication with Selendroid
  --chromedriver-port CHROMEDRIVERPORT
                   Port upon which ChromeDriver will run
  --chromedriver-executable CHROMEDRIVEREXECUTABLE
                   ChromeDriver executable full path
  --use-keystore       (Android-only) When set the keystore will be used to
                   sign apks.
  --keystore-path KEYSTOREPATH
                   (Android-only) Path to keystore
  --keystore-password KEYSTOREPASSWORD
                   (Android-only) Password to keystore
  --key-alias KEYALIAS  (Android-only) Key alias
  --key-password KEYPASSWORD
                   (Android-only) Key password
  --show-config        Show info about the appium server configuration and
                   exit
  --no-perms-check      Bypass Appium's checks to ensure we can read/write
                   necessary files
  --command-timeout DEFAULTCOMMANDTIMEOUT
                   The default command timeout for the server to use for
                   all sessions. Will still be overridden by
```

```
                          newCommandTimeout cap
  --keep-keychains           (iOS) Whether to keep keychains (Library/Keychains)
                          when reset app between sessions
  --strict-caps             Cause sessions to fail if desired caps are sent in
                          that Appium does not recognize as valid for the
                          selected device
  --isolate-sim-device  Xcode 6 has a bug on some platforms where a certain
                          simulator can only be launched without error if all
                          other simulator devices are first deleted. This
                          option causes Appium to delete all devices other than
                          the one being used by Appium. Note that this is a
                          permanent deletion, and you are responsible for using
                          simctl or xcode to manage the categories of devices
                          used with Appium.
  --tmp TMPDIR              Absolute path to directory Appium can use to manage
                          temporary files, like built-in iOS apps it needs to
                          move around. On *nix/Mac defaults to /tmp, on Windows
                          defaults to C:\Windows\Temp
  --trace-dir TRACEDIR  Absolute path to directory Appium use to save ios
                          instruments traces, defaults to <tmp
                          dir>/appium-instruments
  --intent-action INTENTACTION
                          (Android-only) Intent action which will be used to
                          start activity
  --intent-category INTENTCATEGORY
                          (Android-only) Intent category which will be used to
                          start activity
  --intent-flags INTENTFLAGS
                          (Android-only) Flags that will be used to start
                          activity
  --intent-args OPTIONALINTENTARGUMENTS
                          (Android-only) Additional intent arguments that will
                          be used to start activity
  --dont-stop-app-on-reset
                          (Android-only) When included, refrains from stopping
                          the app before restart
  --debug-log-spacing   Add exaggerated spacing in logs to help with visual
                          inspection
  --suppress-adb-kill-server
                          (Android-only) If set, prevents Appium from killing
                          the adb server instance
  --async-trace             Add long stack traces to log entries. Recommended for
                          debugging only.
```

表 4-1-1 中给出了 Appium 服务相关参数的描述。

表 4-1-1　Appium 服务相关参数的说明

参数	描述
automationName	自动化测试引擎的名称，比如 Appium（默认）或 Selendroid
platformName	待测试的手机操作系统，比如 iOS、Android 或 FirefoxOS
platformVersion	手机操作系统版本

（续表）

参数	描述
deviceName	手机 device 或模拟器的 device。在 Android 上，可以通过 adb devices 来得到；在 iOS 上，可以使用 instruments -s devices 来得到
app	.ipa or .apk 文件所在的本地绝对路径或者远程路径
browserName	待自动化测试的手机的 Web 浏览器名称，如果是对 APP 应用进行自动化测试，那么这个关键字的值应该为空
newCommandTimeout	执行命令超时时间，单位为秒。如果达到超时时间仍未接收到新的命令，那么 Appium 会认为客户端退出，然后自动结束会话
autoLaunch	Appium 是否需要自动安装和启动应用，默认为 True
language	设定模拟器（simulator / emulator）的语言（Sim/Emu-only）
locale	设定模拟器（simulator / emulator）的区域设置（Sim/Emu-only）
noReset	不在会话前重置应用状态，默认值为 False
fullReset	Android 上通过卸载而不是清空数据重置应用状态，在 Android 上会话结束后自动清除被测应用，在 iOS 上会删除整个模拟器目录
udid	连接的物理设备的唯一设备标识
orientation	在一个设定的方向模式中开始测试（Sim/Emu-only）

4.2 Appium Library 库的使用

在使用 Appium Library 库时，需要预先安装好 Appium 自动化工具。Appium 官网地址为 http://appium.io/，如图 4-2-1 所示。

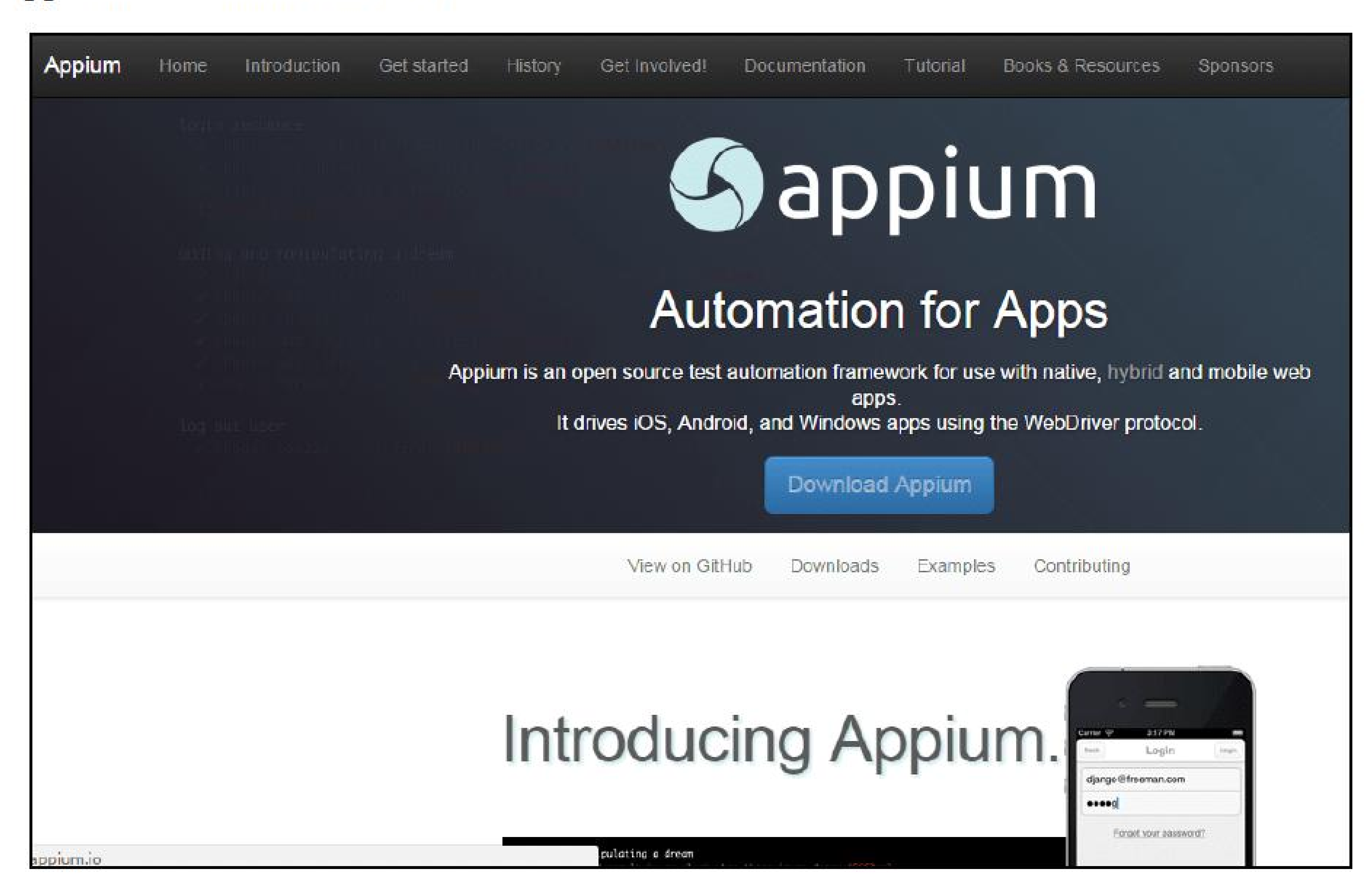


图 4-2-1

Appium 的 GitHub 地址为 https://github.com/appium，如图 4-2-2 所示。

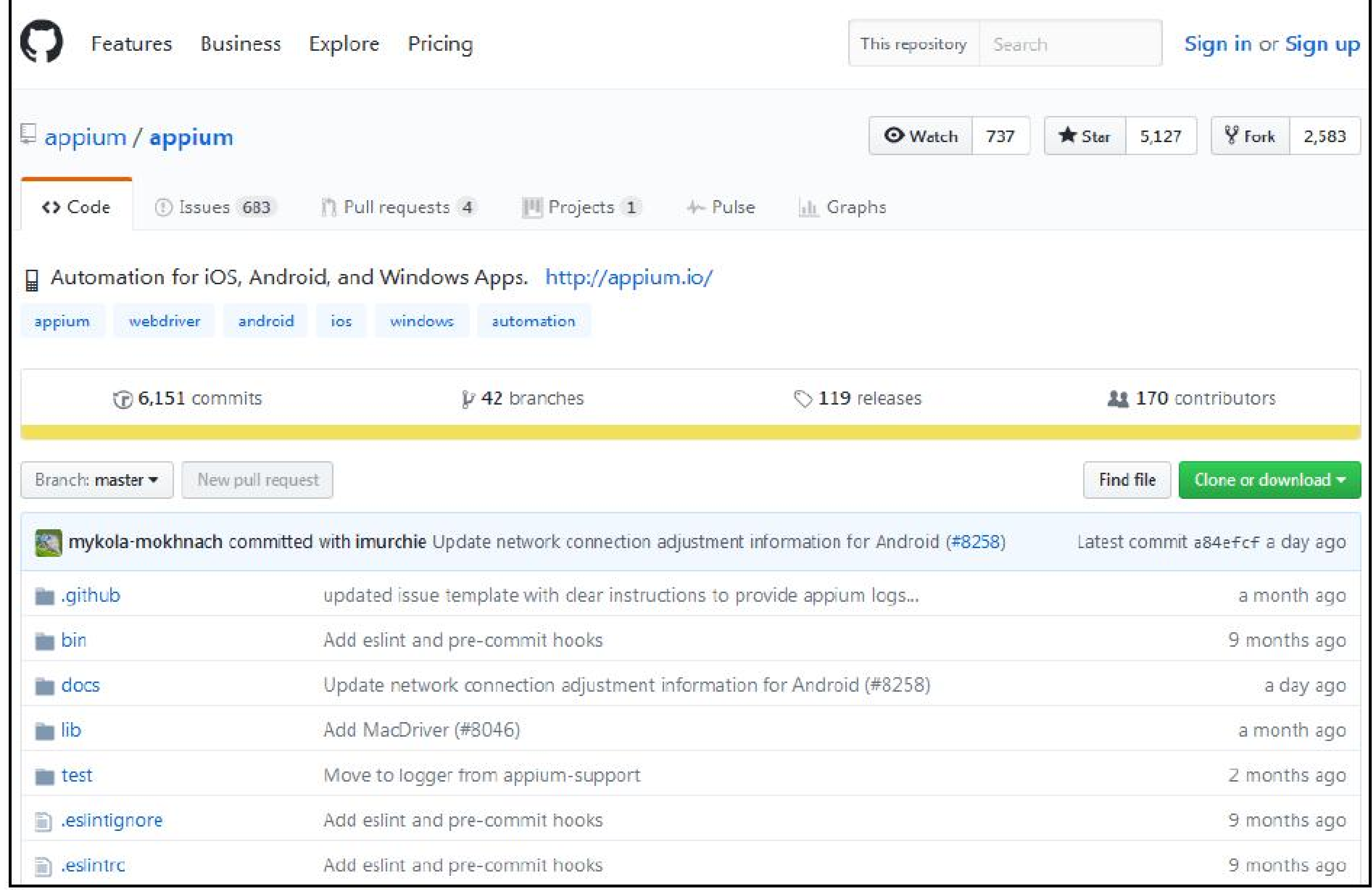


图 4-2-2

Appium 工具提供的 RobotFramework-appiumlibrary 库的访问地址为 https://github.com/serhatbolsu/robotframework-appiumlibrary，如图 4-2-3 所示。可以通过 pip 在线安装 Library，也可以在下载好 Library 库后采用“python setup.py install”的方式来进行安装。

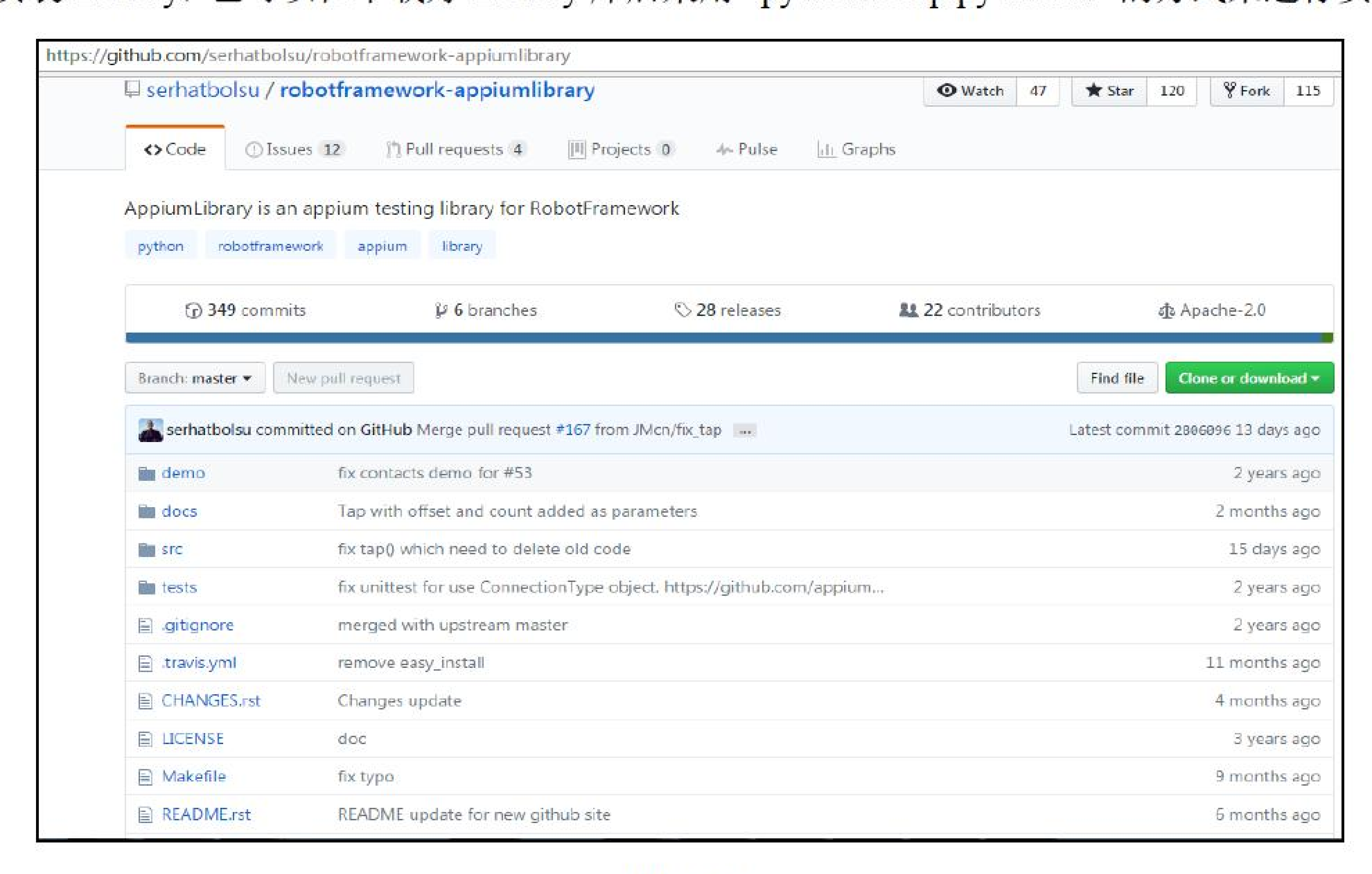


图 4-2-3

- 使用 pip 安装

```
pip install robotframework-appiumlibrary
```

- 使用 setup.py 安装

```
git clone https://github.com/jollychang/robotframework-appiumlibrary.git
cd robotframework-appiumlibrary
python setup.py install
```

4.2.1 Open Application

在 AppiumLibrary 库中，Open Application 关键字用来打开一个待测试移动 APP。

【示例】连接本机已经打开的 appium 服务端，打开一个待测试的安卓 APP，指定测试平台为 Android，测试的手机 deviceName 为 98YFBP522VSU，需要打开的 APP 路径为 C:/Users/yongqing/Desktop/app-debug.apk，APP 的包名为 com.example.calculator，启动的 appActivity 为 MainActivity。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
platformVersion=22 deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
```

执行结果如图 4-2-4 所示。

KEYWORD AppiumLibrary .Open Application http://localhost:4723/wd/hub, platformName=Android, platformVersion=22, deviceName=98YFBP522VSU app=C:/Users/yongqing/Desktop/app-debug.apk, appPackage=com.example.calculator, appActivity=MainActivity

Documentation: Opens a new application to given Appium server.

Start / End / Elapsed: 20170415 22:42:00.863 / 20170415 22:42:14.939 / 00:00:14.076

图 4-2-4

执行完成后，在手机上会自动打开指定的 APP，如图 4-2-5 所示。



图 4-2-5

Appium 端会打印如下运行日志：

```
> info: Found device 98YFBP522VSU
> info: [debug] Setting device id to 98YFBP522VSU
> info: [debug] Waiting for device to be ready and to respond to shell commands
(timeout = 5)
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
```

```
 98YFBP522VSU wait-for-device
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "echo 'ready'"
> info: [debug] Starting logcat capture
> info: [debug] Getting device API level
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "getprop ro.build.version.sdk"
> info: [debug] Device is at API Level 22
> info: Device API level is: 22
> info: [debug] Extracting strings for language: default
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "getprop persist.sys.language"
> info: [debug] Current device persist.sys.language:
> info: [debug] java -jar
"F:\selenium\appium\node_modules\appium\node_modules\appium-adb\jars\appium_a
pk_tools.jar" "stringsFromApk" "C:\Users\yongqing\Desktop\app-debug.apk"
"C:\Users\yongqing\AppData\Local\Temp\com.example.calculator"
> info: [debug] Reading strings from converted strings.json
> info: [debug] Setting language to default
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU push
"C:\\Users\\yongqing\\AppData\\Local\\Temp\\com.example.calculator\\strings.j
son" /data/local/tmp
> info: [debug] Checking whether aapt is present
> info: [debug] Using aapt from
E:\android-sdk-windows\build-tools\24.0.1\aapt.exe
> info: [debug] Retrieving process from manifest.
> info: [debug] executing cmd:
E:\android-sdk-windows\build-tools\24.0.1\aapt.exe dump xmltree
C:\Users\yongqing\Desktop\app-debug.apk AndroidManifest.xml
> info: [debug] Set app process to: com.example.calculator
> info: [debug] Not uninstalling app since server not started with --full-reset
> info: [debug] Checking app cert for C:\Users\yongqing\Desktop\app-debug.apk.
> info: [debug] executing cmd: java -jar
F:\selenium\appium\node_modules\appium\node_modules\appium-adb\jars\verify.ja
r C:\Users\yongqing\Desktop\app-debug.apk
> info: [debug] App already signed.
> info: [debug] Zip-aligning C:\Users\yongqing\Desktop\app-debug.apk
> info: [debug] Checking whether zipalign is present
> info: [debug] Using zipalign from
E:\android-sdk-windows\build-tools\24.0.1\zipalign.exe
> info: [debug] Zip-aligning apk.
> info: [debug] executing cmd:
E:\android-sdk-windows\build-tools\24.0.1\zipalign.exe -f 4
C:\Users\yongqing\Desktop\app-debug.apk
C:\Users\yongqing\AppData\Local\Temp\117315-3596-1p9lthv\appium.tmp
> info: [debug] MD5 for app is 6192e720723dd8700a640a5fb7c59cd2
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "ls /data/local/tmp/6192e720723dd8700a640a5fb7c59cd2.apk"
> info: [debug] Getting install status for com.example.calculator
```

```
> info: [debug] Getting device API level
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "getprop ro.build.version.sdk"
> info: [debug] Device is at API Level 22
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "pm list packages -3 com.example.calculator"
> info: [debug] App is installed
> info: App is already installed, resetting app
> info: [debug] Running fast reset (stop and clear)
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "am force-stop com.example.calculator"
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "pm clear com.example.calculator"
> info: [debug] Forwarding system:4724 to device:4724
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU forward tcp:4724 tcp:4724
> info: [debug] Pushing appium bootstrap to device...
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU push
"F:\\selenium\\appium\\node_modules\\appium\\build\\android_bootstrap\\Appium
Bootstrap.jar" /data/local/tmp/
> info: [debug] Pushing settings apk to device...
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU install
"F:\selenium\appium\node_modules\appium\build\settings_apk\settings_apk-debug
.apk"
> info: [debug] Pushing unlock helper app to device...
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU install
"F:\selenium\appium\node_modules\appium\build\unlock_apk\unlock_apk-debug.apk
"
> info: Starting App
> info: [debug] Attempting to kill all 'uiautomator' processes
> info: [debug] Getting all processes with 'uiautomator'
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "ps 'uiautomator'"
> info: [debug] No matching processes found
> info: [debug] Running bootstrap
> info: [debug] spawning: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell uiautomator runtest AppiumBootstrap.jar -c
io.appium.android.bootstrap.Bootstrap -e pkg com.example.calculator -e
disableAndroidWatchers false
> info: [debug] [UIAUTOMATOR STDOUT] INSTRUMENTATION_STATUS: numtests=1
> info: [debug] [UIAUTOMATOR STDOUT] INSTRUMENTATION_STATUS: stream=
> info: [debug] [UIAUTOMATOR STDOUT] io.appium.android.bootstrap.Bootstrap:
> info: [debug] [UIAUTOMATOR STDOUT] INSTRUMENTATION_STATUS:
id=UiAutomatorTestRunner
> info: [debug] [UIAUTOMATOR STDOUT] INSTRUMENTATION_STATUS: test=testRunServer
> info: [debug] [UIAUTOMATOR STDOUT] INSTRUMENTATION_STATUS:
class=io.appium.android.bootstrap.Bootstrap
```

```
> info: [debug] [UIAUTOMATOR STDOUT] INSTRUMENTATION_STATUS: current=1
> info: [debug] [UIAUTOMATOR STDOUT] INSTRUMENTATION_STATUS_CODE: 1
> info: [debug] [BOOTSTRAP] [debug] Socket opened on port 4724
> info: [debug] [BOOTSTRAP] [debug] Appium Socket Server Ready
> info: [debug] [BOOTSTRAP] [debug] Loading json...
> info: [debug] [BOOTSTRAP] [debug] json loading complete.
> info: [debug] Waking up device if it's not alive
> info: [debug] Pushing command to appium work queue: ["wake",{}]
> info: [debug] [BOOTSTRAP] [debug] Registered crash watchers.
> info: [debug] [BOOTSTRAP] [debug] Client connected
> info: [debug] [BOOTSTRAP] [debug] Got data from client:
{"cmd":"action","action":"wake","params":{}}
> info: [debug] [BOOTSTRAP] [debug] Got command of type ACTION
> info: [debug] [BOOTSTRAP] [debug] Got command action: wake
> info: [debug] [BOOTSTRAP] [debug] Returning result: {"status":0,"value":true}
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "dumpsys window"
> info: [debug] Screen already unlocked, continuing.
> info: [debug] Pushing command to appium work queue: ["getDataDir",{}]
> info: [debug] [BOOTSTRAP] [debug] Got data from client:
{"cmd":"action","action":"getDataDir","params":{}}
> info: [debug] [BOOTSTRAP] [debug] Got command of type ACTION
> info: [debug] [BOOTSTRAP] [debug] Got command action: getDataDir
> info: [debug] [BOOTSTRAP] [debug] Returning result:
{"status":0,"value":"\/data\/local\/tmp"}
> info: [debug] dataDir set to: /data/local/tmp
> info: [debug] Pushing command to appium work queue:
["compressedLayoutHierarchy",{"compressLayout":false}]
> info: [debug] [BOOTSTRAP] [debug] Got data from client:
{"cmd":"action","action":"compressedLayoutHierarchy","params":{"compressLayou
t":false}}
> info: [debug] [BOOTSTRAP] [debug] Got command of type ACTION
> info: [debug] [BOOTSTRAP] [debug] Got command action: compressedLayoutHierarchy
> info: [debug] [BOOTSTRAP] [debug] Returning result: {"status":0,"value":false}
> info: [debug] Getting device API level
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "getprop ro.build.version.sdk"
> info: [debug] Device is at API Level 22
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "am start -S -a android.intent.action.MAIN -c
android.intent.category.LAUNCHER -f 0x10200000 -n
com.example.calculator/MainActivity"
> info: [debug] We tried to start an activity that doesn't exist, retrying with .
prepended to activity
> info: [debug] Getting device API level
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "getprop ro.build.version.sdk"
> info: [debug] Device is at API Level 22
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "am start -S -a android.intent.action.MAIN -c
```

```
android.intent.category.LAUNCHER -f 0x10200000 -n
com.example.calculator/.MainActivity"
> info: [debug] Waiting for pkg "com.example.calculator" and activity
"MainActivity" to be focused
> info: [debug] Getting focused package and activity
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "dumpsys window windows"
> info: [debug] executing cmd: E:\android-sdk-windows\platform-tools\adb.exe -s
98YFBP522VSU shell "getprop ro.build.version.release"
> info: [debug] Device is at release version 5.1
> info: [debug] Device launched! Ready for commands
> info: [debug] Setting command timeout to the default of 60 secs
> info: [debug] Appium session started with sessionId
090a2542-63ee-4e1b-912f-32d67922d49e
> info: <-- POST /wd/hub/session 303 13662.949 ms - 74
> info: --> GET /wd/hub/session/090a2542-63ee-4e1b-912f-32d67922d49e {}
> info: [debug] Responding to client with success:
{"status":0,"value":{"platform":"LINUX","browserName":"Android","platformVers
ion":"5.1","webStorageEnabled":false,"takesScreenshot":true,"javascriptEnable
d":true,"databaseEnabled":false,"networkConnectionEnabled":true,"locationCont
extEnabled":false,"warnings":{},"desired":{"deviceName":"98YFBP522VSU","app":
"C:/Users/yongqing/Desktop/app-debug.apk","platformVersion":"22","appPackage"
:"com.example.calculator","platformName":"Android","appActivity":"MainActivit
y"},"deviceName":"98YFBP522VSU","app":"C:/Users/yongqing/Desktop/app-debug.ap
k","appPackage":"com.example.calculator","platformName":"Android","appActivit
y":"MainActivity"},"sessionId":"090a2542-63ee-4e1b-912f-32d67922d49e"}
> info: <-- GET /wd/hub/session/090a2542-63ee-4e1b-912f-32d67922d49e 200 2.395
ms - 686
{"status":0,"value":{"platform":"LINUX","browserName":"Android","platformVers
ion":"5.1","webStorageEnabled":false,"takesScreenshot":true,"javascriptEnable
d":true,"databaseEnabled":false,"networkConnectionEnabled":true,"locationCont
extEnabled":false,"warnings":{},"desired":{"deviceName":"98YFBP522VSU","app":
"C:/Users/yongqing/Desktop/app-debug.apk","platformVersion":"22","appPackage"
:"com.example.calculator","platformName":"Android","appActivity":"MainActivit
y"},"deviceName":"98YFBP522VSU","app":"C:/Users/yongqing/Desktop/app-debug.ap
k","appPackage":"com.example.calculator","platformName":"Android","appActivit
y":"MainActivity"},"sessionId":"090a2542-63ee-4e1b-912f-32d67922d49e"}
```

4.2.2　Input Text 和 Click Button

Input Text 关键字一般用来给输入框进行输入操作，接收[locator | text]两个参数。

【示例 1】启动安卓手机上一个 APP 的 MainActivity，在打开 Activity 进入界面后，分别向两个 EditText 输入框中输入“12”，并且单击 “计算”按钮来计算输入的两个数字的乘积。

APP 的界面，提供了两个输入框、一个计算按钮（Button），如图 4-2-6 所示。



图 4-2-6

在写这个自动化案例前，我们可以使用安卓 SDK 提供的 Ui Automator Viewer 工具来进行这个界面的资源定位。通过定位，可以看到第一个 EditText 输入框的 resource-id 为“com.example.calculator:id/factorone”、class 为“android.widget.EditText”、text 为“请输入数字”，如图 4-2-7 所示。

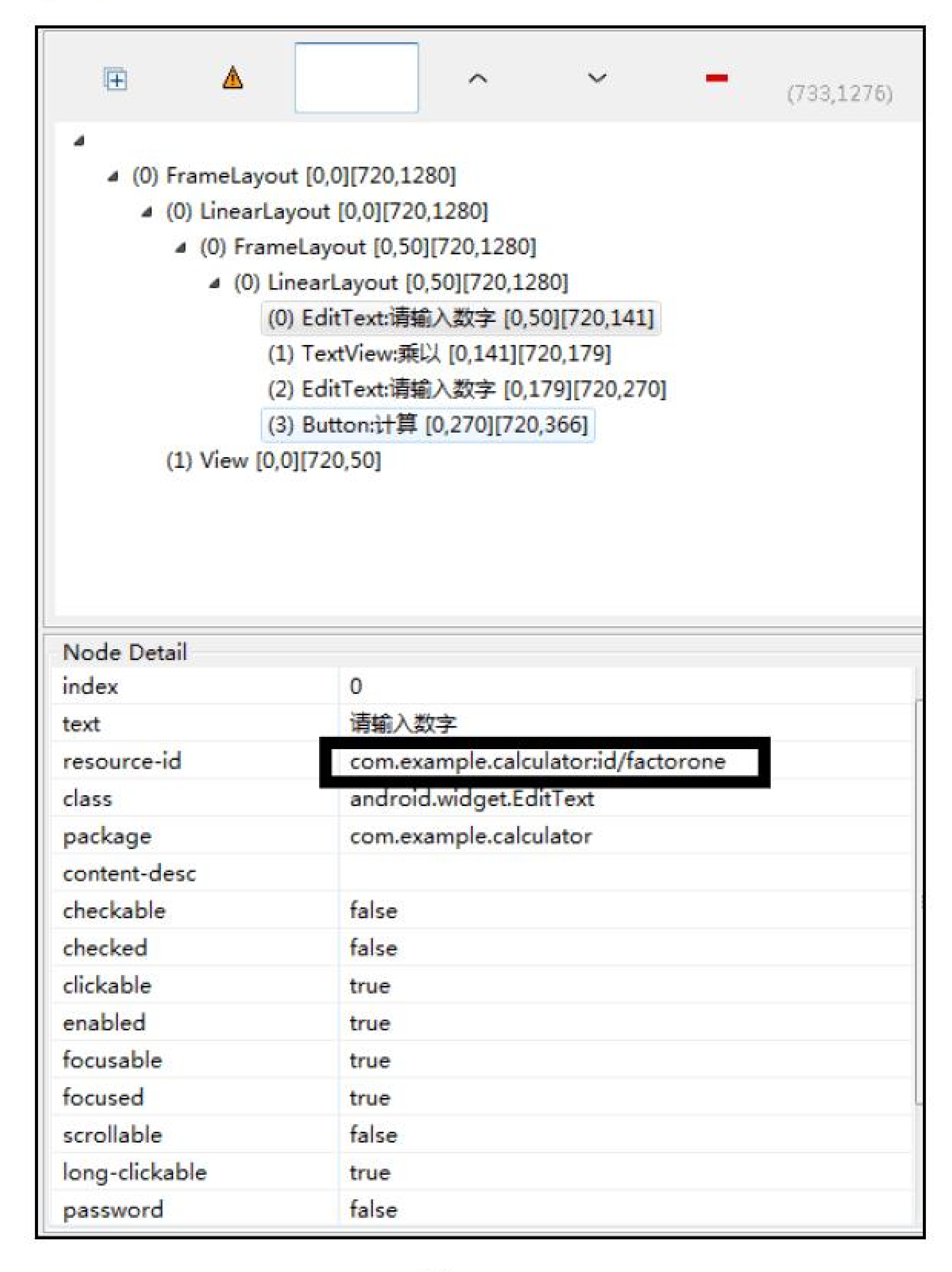


图 4-2-7

第二个 EditText 输入框的 resource-id 为“com.example.calculator:id/factortwo”、class 为“android.widget.EditText”、text 为“请输入数字”，如图 4-2-8 所示。

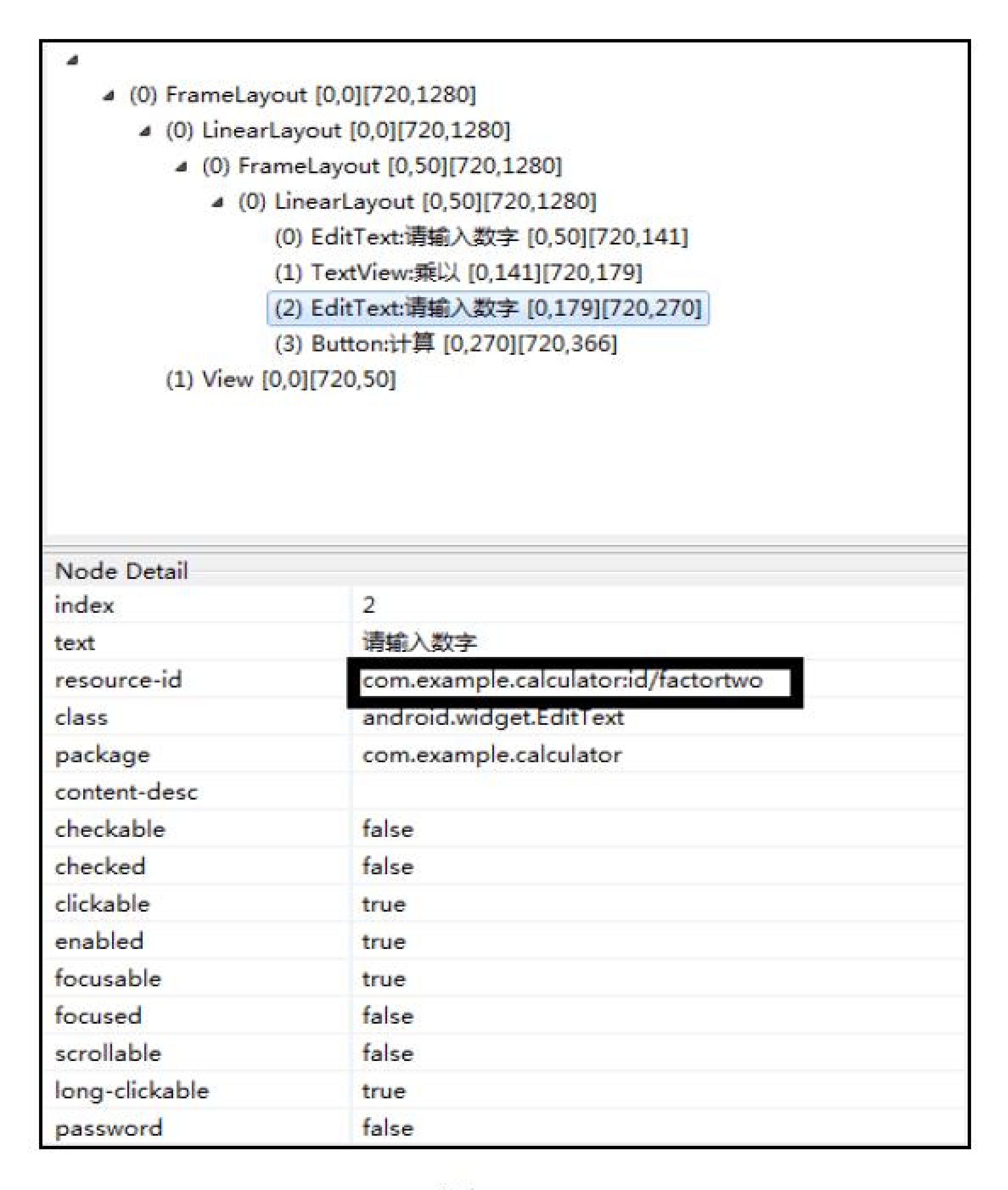


图 4-2-8

Button 按钮的 resource-id 为“com.example.calculator:id/commit”、class 为“android.widget.Button”、text 为“计算”，如图 4-2-9 所示。

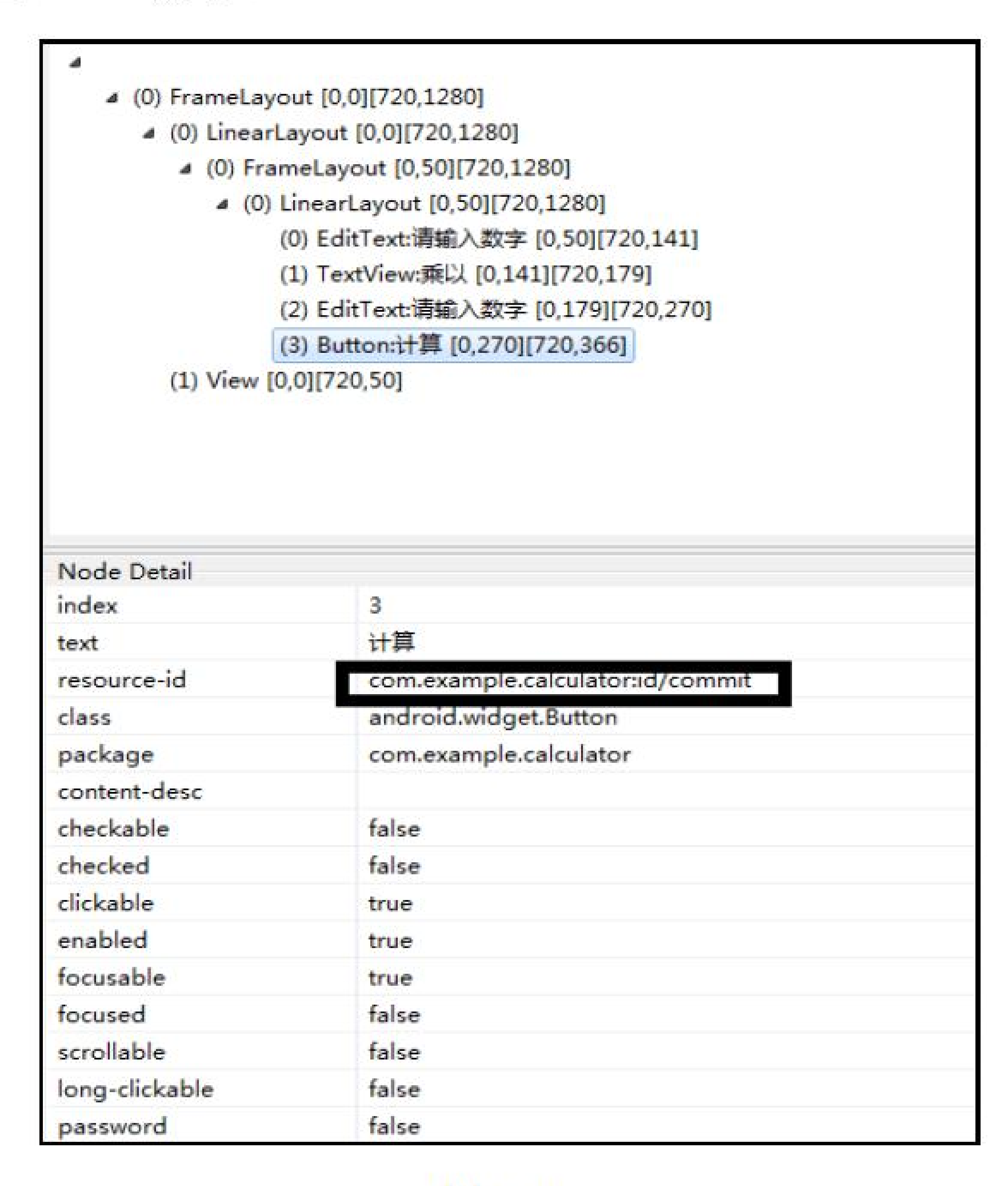


图 4-2-9

Click Button 关键字用来模拟单击 APP 上的一个 button 按钮，该关键字接收一个参数 [index_or_name]。

```
Open Application  http://localhost:4723/wd/hub   platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Input Text    id=com.example.calculator:id/factorone   12
Input Text    id=com.example.calculator:id/factortwo   12
Click Button  计算
```

执行结果如图 4-2-10 所示。

```
KEYWORD AppiumLibrary.Open Application http://localhost:4723/wd/hub, platformName=Android, platformVersion=22, deviceName=98YFBP522VSU, app=C:/Users/yongqing/Desktop/app-debug.apk, appPackage=com.example.calculator, appActivity=MainActivity
Documentation:        Opens a new application to given Appium server.
Start / End / Elapsed: 20170429 16:07:13.080 / 20170429 16:07:28.442 / 00:00:15.362
KEYWORD AppiumLibrary.Input Text id=factorone, 12
Documentation:        Types the given `text` into text field identified by `locator`.
Start / End / Elapsed: 20170429 16:07:28.444 / 20170429 16:07:33.265 / 00:00:04.821
16:07:28.446  INFO  Typing text '12' into text field 'id=factorone'
KEYWORD AppiumLibrary.Input Text id=factortwo, 12
Documentation:        Types the given `text` into text field identified by `locator`.
Start / End / Elapsed: 20170429 16:07:33.266 / 20170429 16:07:38.573 / 00:00:05.307
16:07:33.266  INFO  Typing text '12' into text field 'id=factortwo'
KEYWORD AppiumLibrary.Click Button 计算
Documentation:        Click button
Start / End / Elapsed: 20170429 16:07:38.583 / 20170429 16:07:39.683 / 00:00:01.100
16:07:39.164  INFO  '计算'.
16:07:39.235  INFO  Clicking element '计算'.
```

图 4-2-10

可以看到已经执行成功。上面通过 resource-id 的方式来定位 EditText 输入框，并且通过 name 的方式来定位 button 按钮。

下面使用另一种方式，即通过 name 的方式来定位 EditText 输入框、通过 index 的方式来单击 button 按钮。

```
Open Application   http://localhost:4723/wd/hub   platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator
Input Text  name=请输入数字   12
Input Text  name=请输入数字   14
Click Button    index=0
```

执行结果如图 4-2-11 所示。

```
- KEYWORD AppiumLibrary.Open Application http://localhost:4723/wd/hub, platformName=Android, platformVersion=22, deviceName=98YFBP522VSU,
  app=C:/Users/yongqing/Desktop/app-debug.apk, appPackage=com.example.calculator, appActivity=MainActivity
  Documentation:       Opens a new application to given Appium server.
  Start / End / Elapsed:   20170501 00:41:43.641 / 20170501 00:41:55.900 / 00:00:12.259
- KEYWORD AppiumLibrary.Input Text name=请输入数字, 12
  Documentation:       Types the given `text` into text field identified by `locator`.
  Start / End / Elapsed:   20170501 00:41:55.901 / 20170501 00:42:01.244 / 00:00:05.343
  00:41:55.902   INFO   Typing text '12' into text field 'name=请输入数字'
- KEYWORD AppiumLibrary.Input Text name=请输入数字, 14
  Documentation:       Types the given `text` into text field identified by `locator`.
  Start / End / Elapsed:   20170501 00:42:01.245 / 20170501 00:42:06.597 / 00:00:05.352
  00:42:01.246   INFO   Typing text '14' into text field 'name=请输入数字'
- KEYWORD AppiumLibrary.Click Button index=0
  Documentation:       Click button
  Start / End / Elapsed:   20170501 00:42:06.597 / 20170501 00:42:07.635 / 00:00:01.038
  00:42:07.261   INFO   Clicking element '计算'.
  00:42:07.634   INFO   elements:"[<appium.webdriver.webelement.WebElement (session="0fe4746b-802f-49f3-99d9-cedfd547bd38", element="4")>]"
                        index:"0"
                        element: <appium.webdriver.webelement.WebElement (session="0fe4746b-802f-49f3-99d9-cedfd547bd38", element="4")>
```

图 4-2-11

在通过 index 的方式来单击 button 按钮的时候，注意 index 的取值不要和通过 Ui Automator Viewer 工具看到的 index 混淆。先看一段 AppiumLibrary 库的源码，在这里选取了源码中的三个函数。从如下三个函数中可以看到 click_button 关键字支持 name 和 index 两种方式来定位一个 button。在使用 index 的时候，是根据 class_name，即通过 android.widget.Button 这个 class_name 来找出当前界面中有几个 button 按钮（源码中通过 elements = self._find_elements_by_class_name(class_name)来寻找，有几个 button 按钮，就会返回几个 element），然后每个 button 按钮按照 index 的方式来取出（源码中通过 element = elements[index] 来得到具体的一个 button 按钮）。

AppiumLibrary 库函数 1：

```
def click_button(self, index_or_name):
    """ Click button """
    _platform_class_dict = {'ios': 'UIAButton',
                            'android': 'android.widget.Button'}
    if self._is_support_platform(_platform_class_dict):
        class_name = self._get_class(_platform_class_dict)
        self._click_element_by_class_name(class_name, index_or_name)
```

AppiumLibrary 库函数 2：

```
def _click_element_by_class_name(self, class_name, index_or_name):
        element = self._find_element_by_class_name(class_name, index_or_name)
        self._info("Clicking element '%s'." % element.text)
        try:
            element.click()
        except Exception as e:
            raise 'Cannot click the %s element "%s"' % (class_name, index_or_name)
```

AppiumLibrary 库函数 3：

```
def _find_element_by_class_name(self, class_name, index_or_name):
    elements = self._find_elements_by_class_name(class_name)
```

```
    print 'elements:"%s"' % elements

    if self._is_index(index_or_name):
        try:
            index = int(index_or_name.split('=')[-1])
            print 'index:"%s"' % index
            element = elements[index]
            print 'element:' , element
        except (IndexError, TypeError):
            raise 'Cannot find the element with index "%s"' % index_or_name
    else:
        found = False
        for element in elements:
            self._info("'%s'." % element.text)
            if element.text == index_or_name:
                found = True
                break
        if not found:
            raise 'Cannot find the element with name "%s"' % index_or_name
```

在界面中放入两个 button 按钮，一个 button 按钮是“计算”按钮，一个 button 按钮是“取消”按钮，如图 4-2-12 所示。

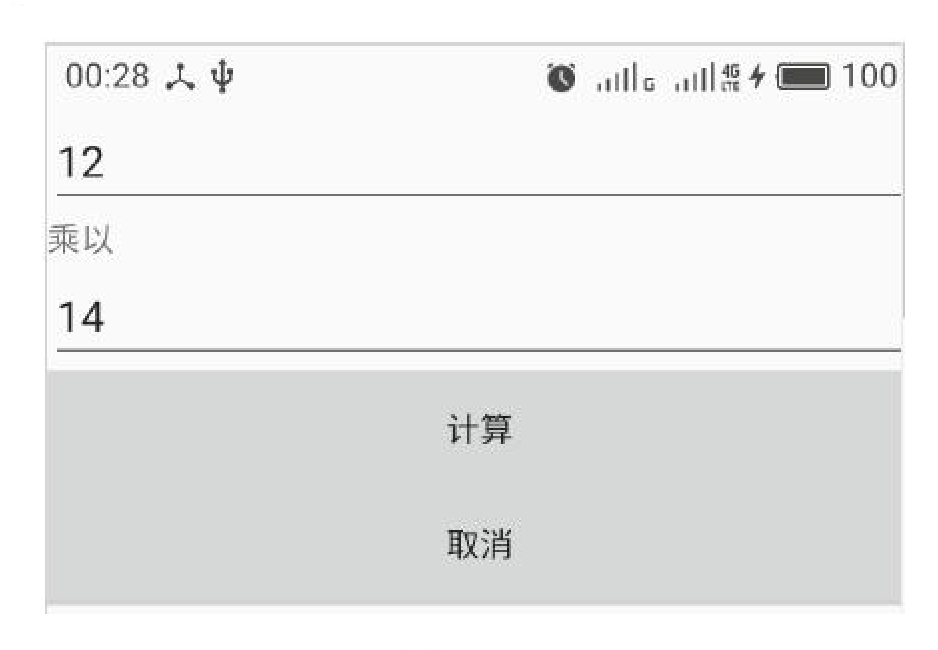


图 4-2-12

在执行时，通过 elements = self._find_elements_by_class_name(class_name)得到所有的 button 按钮后，再用 print 'elements:"%s"' % elements 打印出获取到的 elements，也就是所有的 button。从如下的输出结果可以看到 elements 会存放在一个列表中，该列表中共有两个元素，代表取到了两个 button 按钮：

```
elements:"[<appium.webdriver.webelement.WebElement
(session="8e85c12f-2243-4b82-abfc-d091fddbed8b", element="4")>,
<appium.webdriver.webelement.WebElement
(session="8e85c12f-2243-4b82-abfc-d091fddbed8b", element="5")>]"
```

当 index 为 0 时，会取到第一个按钮，也就是“计算”这个 button 按钮；当 index 为 1 时，会取到第二个按钮，也就是“取消”这个 button 按钮；当 index 超过 1 后，就会报错，此时源码中会通过 raise *'Cannot find the element with index "%s"'* % index_or_name 来抛出一个异常，告诉使用者，不能通过当前的 index 获取到 element（也就是此时无法获取到任何 button 按钮）。

【示例 2】通过 xpath 的方式定位元素，这里依旧以上面的 APP 界面为示例。

用 xpath 的方式定位第一个 EditText 输入框和第二个 EditText 输入框，示例如下：

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Input Text  xpath=//android.widget.EditText[1]   12
Input Text  xpath=//android.widget.EditText[2]   14
Click Button    计算
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase005
20170510 13:45:07.381 :  INFO : Typing text '12' into text field
'xpath=//android.widget.EditText[1]'
20170510 13:45:07.381 :  INFO : msg:find xpath=//android.widget.EditText[1]
20170510 13:45:07.381 :  INFO : prefix: xpath
20170510 13:45:07.397 :  INFO : criteria: //android.widget.EditText[1]
20170510 13:45:10.462 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="ec48b38a-9cbe-457d-94a0-dec662d3f9cb", element="1")>]
20170510 13:45:15.313 :  INFO : Typing text '14' into text field
'xpath=//android.widget.EditText[2]'
20170510 13:45:15.313 :  INFO : msg:find xpath=//android.widget.EditText[2]
20170510 13:45:15.313 :  INFO : prefix: xpath
20170510 13:45:15.313 :  INFO : criteria: //android.widget.EditText[2]
20170510 13:45:15.906 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="ec48b38a-9cbe-457d-94a0-dec662d3f9cb", element="2")>]
20170510 13:45:21.307 :  INFO : '计算'.
20170510 13:45:21.385 :  INFO : Clicking element '计算'.
Ending test:  RobotFrameworkTest1.TestSuite5.TestCase005
```

从上面的执行结果看，通过 xpath=//android.widget.EditText[1] 定位到了第一个输入框，通过 xpath=//android.widget.EditText[2] 定位到了第二个输入框。

【示例 3】通过 accessibility_id 的方式定位元素。accessibility_id 对应到安卓 APP 后，其对应的属性为 content-desc，这里依旧以上面的 APP 界面为示例，但是我们对第一个 EditText 输入框加入了 content-desc 属性，如图 4-2-13 所示。

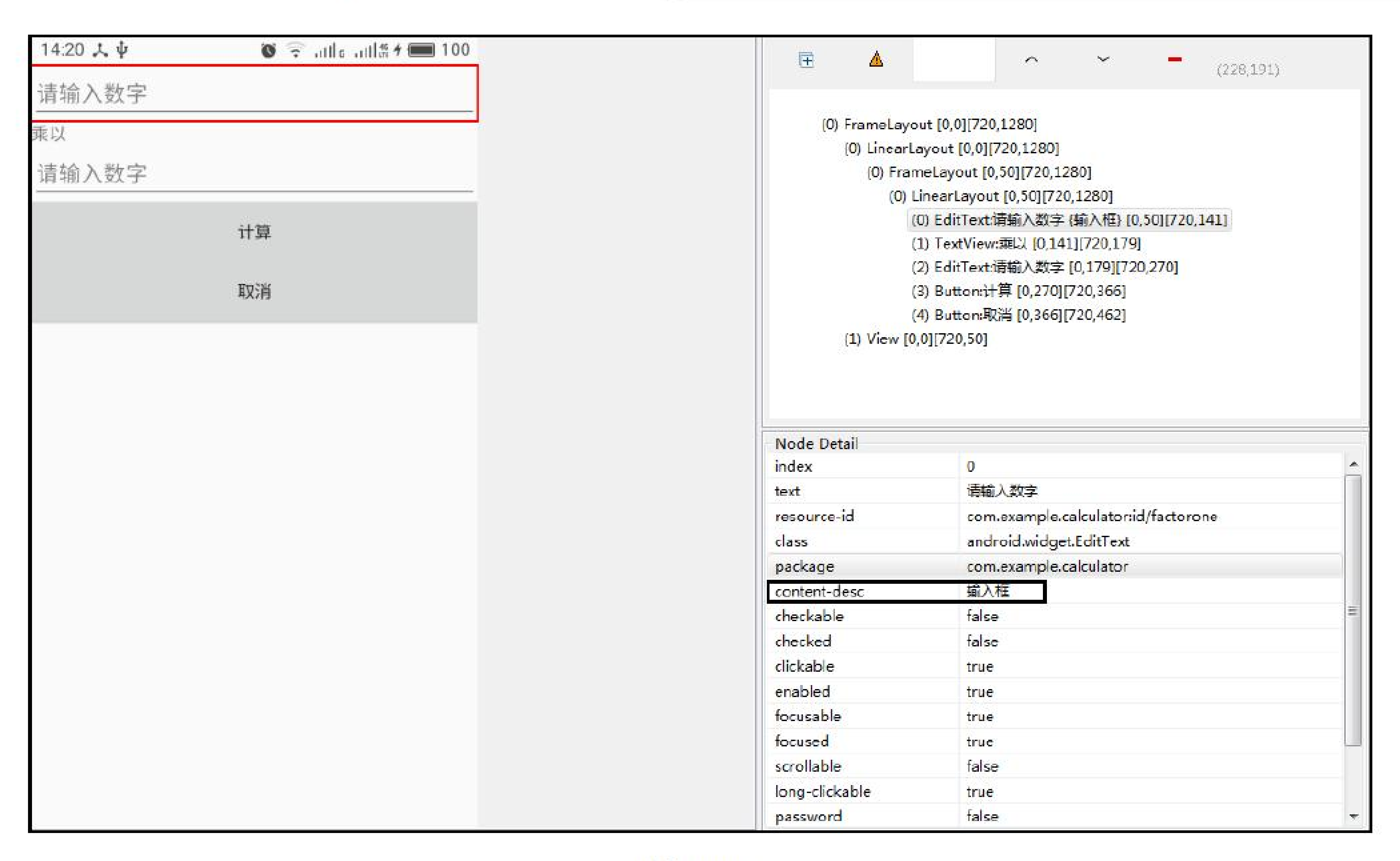


图 4-2-13

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Input Text  accessibility_id=输入框   23
Input Text  id=com.example.calculator:id/factortwo 12
Click Button    计算
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase006
20170510 14:23:09.735 :  INFO : Typing text '23' into text field 'accessibility_id=
输入框'
20170510 14:23:09.735 :  INFO : msg:find accessibility_id=输入框
20170510 14:23:16.573 :  INFO : Typing text '12' into text field
'id=com.example.calculator:id/factortwo'
20170510 14:23:16.573 :  INFO : msg:find id=com.example.calculator:id/factortwo
20170510 14:23:22.799 :  INFO : '计算'.
20170510 14:23:22.901 :  INFO : Clicking element '计算'.
Ending test:  RobotFrameworkTest1.TestSuite5.TestCase006
```

从执行结果看，通过“accessibility_id=输入框”也可以定位到 EditText 输入框。

4.2.3 Clear Text

Clear Text 关键字用来清除输入框的数据，接收一个参数[locator]，这里的 locator 指的就是界面元素的定位方式。

【示例 1】Clear Text 清除输入框数据时，采用 resource-id 的方式来定位输入框。这里依旧采用上面使用的 APP 界面来做操作示例，将输入的数字 12 通过 Clear Text id=com.example.calculator:id/factorone 来清除。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Input Text  name=请输入数字   12
Input Text  name=请输入数字   14
Click Button    index=1
Clear Text  id=com.example.calculator:id/factorone
```

执行结果如图 4-2-14 所示。

```
KEYWORD AppiumLibrary.Open Application http://localhost:4723/wd/hub, platformName=Android, platformVersion=22, deviceName=98YFBP522VSU,
app=C:/Users/yongqing/Desktop/app-debug.apk, appPackage=com.example.calculator, appActivity=MainActivity    00:00:11.347
Documentation:      Opens a new application to given Appium server.
Start / End / Elapsed:  20170502 23:02:11.999 / 20170502 23:02:23.346 / 00:00:11.347
KEYWORD AppiumLibrary.Input Text name=请输入数字, 12    00:00:07.610
Documentation:      Types the given 'text' into text field identified by 'locator'.
Start / End / Elapsed:  20170502 23:02:23.348 / 20170502 23:02:30.958 / 00:00:07.610
23:02:23.349  INFO  Typing text '12' into text field 'name=请输入数字'
KEYWORD AppiumLibrary.Input Text name=请输入数字, 14    00:00:05.382
Documentation:      Types the given 'text' into text field identified by 'locator'.
Start / End / Elapsed:  20170502 23:02:30.959 / 20170502 23:02:36.341 / 00:00:05.382
23:02:30.960  INFO  Typing text '14' into text field 'name=请输入数字'
KEYWORD AppiumLibrary.Click Button index=1    00:00:03.778
Documentation:      Click button
Start / End / Elapsed:  20170502 23:02:36.342 / 20170502 23:02:40.120 / 00:00:03.778
23:02:36.962  INFO  Clicking element '取消'.
23:02:40.119  INFO  elements:"[<appium.webdriver.webelement.WebElement (session="ec1247ad-a8c9-4f4f-a739-daed18a672a2", element="4")>, <appium.webdriver.webelement.WebElement (session="ec1247ad-a8c9-4f4f-a739-daed18a672a2", element="5")>]"
                    index:"1"
                    element: <appium.webdriver.webelement.WebElement (session="ec1247ad-a8c9-4f4f-a739-daed18a672a2", element="5")>
KEYWORD AppiumLibrary.Clear Text id=com.example.calculator:id/factorone    00:00:09.410
Documentation:      Clears the text field identified by 'locator'.
Start / End / Elapsed:  20170502 23:02:40.120 / 20170502 23:02:49.530 / 00:00:09.410
23:02:40.121  INFO  Clear text field 'id=com.example.calculator:id/factorone'
```

图 4-2-14

【示例 2】Clear Text 清除输入框数据时，采用 xpath 的方式来定位输入框，这里依旧采用上面使用的 APP 界面来做操作示例，将输入的数字 12 通过 Clear Text xpath=//android.widget.EditText[1]来清除掉。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Input Text  name=请输入数字   12
Input Text  name=请输入数字   5
Click Button    index=1
Clear Text  xpath=//android.widget.EditText[1]
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase004
20170510 13:53:54.517 :  INFO : Typing text '12' into text field 'name=请输入数
字'
20170510 13:53:54.517 :  INFO : msg:find name=请输入数字
20170510 13:53:54.517 :  INFO : prefix: name
```

```
20170510 13:53:54.517 :  INFO : criteria: 请输入数字
20170510 13:53:56.421 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="b5daf778-7f94-495c-971d-77b0ad5f52ed", element="1")>,
<appium.webdriver.webelement.WebElement
(session="b5daf778-7f94-495c-971d-77b0ad5f52ed", element="2")>]
20170510 13:54:01.366 :  INFO : Typing text '5' into text field 'name=请输入数字
'
20170510 13:54:01.381 :  INFO : msg:find name=请输入数字
20170510 13:54:01.381 :  INFO : prefix: name
20170510 13:54:01.381 :  INFO : criteria: 请输入数字
20170510 13:54:01.927 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="b5daf778-7f94-495c-971d-77b0ad5f52ed", element="3")>]
20170510 13:54:07.416 :  INFO : Clicking element '取消'.
20170510 13:54:10.552 :  INFO : Clear text field
'xpath=//android.widget.EditText[1]'
20170510 13:54:10.552 :  INFO : msg:find xpath=//android.widget.EditText[1]
20170510 13:54:10.552 :  INFO : prefix: xpath
20170510 13:54:10.552 :  INFO : criteria: //android.widget.EditText[1]
20170510 13:54:10.848 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="b5daf778-7f94-495c-971d-77b0ad5f52ed", element="6")>]
20170510 13:54:10.848 :  INFO : execute element.clear by
<appium.webdriver.webelement.WebElement
(session="b5daf778-7f94-495c-971d-77b0ad5f52ed", element="6")>
Ending test:   RobotFrameworkTest1.TestSuite5.TestCase004
```

从上面的执行日志看，通过 xpath=//android.widget.EditText[1]成功定位到了输入框，并且执行 Clear Text 关键字成功。

4.2.4 Click Element

Click Element 关键字用来模拟单击 APP 界面上的一个元素，接收一个参数[locator] ，这里的 locator 指的是界面元素的定位方式。

【示例】使用 Click Element 关键字来模拟单击一个 button 按钮，这里 locator 使用 name 的方式来定位需要单击的元素，依旧采用上面使用的 APP 界面来做操作示例。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Input Text  accessibility_id=输入框  23
Input Text  id=com.example.calculator:id/factortwo 12
Click Element   name=计算
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase007
```

```
20170510 15:13:18.868 :  INFO : Typing text '23' into text field 'accessibility_id=输入框'
20170510 15:13:18.868 :  INFO : msg:find accessibility_id=输入框
20170510 15:13:25.718 :  INFO : msg:find id=com.example.calculator:id/factortwo
20170510 15:13:31.077 :  INFO : Clicking element 'name=计算'.
20170510 15:13:31.077 :  INFO : msg:find name=计算
20170510 15:13:31.704 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="3b92f067-4ddd-4dc5-90eb-c6915eb8e364", element="3")>]
Ending test:  RobotFrameworkTest1.TestSuite5.TestCase007
```

从执行结果可以看到，通过 Click Element 关键字也可以模拟单击一个 button 按钮。

4.2.5　Click A Point

Click A Point 关键字用来模拟单击 APP 界面上的一个点，接收 [x=0 | y=0 | duration=100] 三个参数，x 和 y 代表的是点的坐标位置，duration 代表的是单击持续的时长。这三个参数如果没有传入，就会默认 x=0、y=0、duration=100。

【示例】这里依旧采用上面使用的 APP 界面来做操作示例，使用 Click A Point 关键字来模拟单击一个 button 按钮，输入 button 按钮所在的坐标。

```
Open Application   http://localhost:4723/wd/hub   platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Input Text  accessibility_id=输入框  23
Input Text  id=com.example.calculator:id/factortwo 12
Click A Point   370 339 1000
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase008
20170510 15:32:25.243 :  INFO : Typing text '23' into text field 'accessibility_id=输入框'
20170510 15:32:25.243 :  INFO : msg:find accessibility_id=输入框
20170510 15:32:32.005 :  INFO : Typing text '12' into text field
'id=com.example.calculator:id/factortwo'
20170510 15:32:32.006 :  INFO : msg:find id=com.example.calculator:id/factortwo
20170510 15:32:37.282 :  INFO : Clicking on a point (370,339).
Ending test:  RobotFrameworkTest1.TestSuite5.TestCase008
```

从执行结果看到，还可以通过 Click A Point 关键字模拟单击一个 button 按钮。

4.2.6　Click Element At Coordinates

Click Element At Coordinates 关键字通过一个具体的坐标点来模拟单击一个 Element，接收 [coordinate_X | coordinate_Y]两个参数。

【示例】这里依旧采用上面使用的 APP 界面来做操作示例，使用 Click Element At

Coordinates 关键字来模拟单击一个 button 按钮，输入 button 按钮所在的坐标。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Input Text  accessibility_id=输入框  23
Input Text  id=com.example.calculator:id/factortwo 12
Click Element At Coordinates    370 339
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase009
20170510 15:43:00.061 :  INFO : Typing text '23' into text field 'accessibility_id=
输入框'
20170510 15:43:00.062 :  INFO : msg:find accessibility_id=输入框
20170510 15:43:06.198 :  INFO : Typing text '12' into text field
'id=com.example.calculator:id/factortwo'
20170510 15:43:06.198 :  INFO : msg:find id=com.example.calculator:id/factortwo
20170510 15:43:11.649 :  INFO : Pressing at (370, 339).
Ending test:   RobotFrameworkTest1.TestSuite5.TestCase009
```

从执行结果看到，使用 Click Element At Coordinates 关键字可以成功模拟一个单击按钮的操作。

4.2.7 Get Element Location

Get Element Location 关键字用来获取一个 Element 的 Location 位置，接收一个参数 [locator]。

【示例】使用 Get Element Location 来获取一个 EditText 输入框的 Location（位置），在这里依旧采用上面使用的 APP 界面来做操作示例。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
${Location} Get Element Location    id=com.example.calculator:id/factorone
log ${Location}
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase010
20170510 15:56:05.348 :  INFO : msg:find id=com.example.calculator:id/factorone
20170510 15:56:06.929 :  INFO : Element 'id=com.example.calculator:id/factorone'
location: {'y': 50, 'x': 0}
20170510 15:56:06.930 :  INFO : ${Location} = {'y': 50, 'x': 0}
20170510 15:56:06.932 :  INFO : {'y': 50, 'x': 0}
Ending test:   RobotFrameworkTest1.TestSuite5.TestCase010
```

从执行结果可以看到，获取到的 Location 结果为{'y': 50, 'x': 0}。

4.2.8　Get Current Context

Get Current Context 关键字用于获取当前的上下文，不需要接收任何参数。

【示例】使用 Get Current Context 来获取当前 APP 的上下文，在这里依旧采用上面使用的 APP 来做操作示例。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
${Context}  Get Current Context
log ${Context}
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase011
20170510 16:02:17.889 :  INFO : ${Context} = NATIVE_APP
20170510 16:02:17.891 :  INFO : NATIVE_APP
Ending test:   RobotFrameworkTest1.TestSuite5.TestCase011
```

从执行结果可以看到，获取到的 Context 结果为 NATIVE_APP。

4.2.9　Get Contexts

Get Contexts 关键字用于获取当前所有的上下文，不需要接收任何参数。

【示例】使用 Get Contexts 来获取当前 APP 的所有上下文，在这里依旧采用上面使用的 APP 来做操作示例。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
${Contexts} Get Contexts
log ${Contexts}
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase012
20170510 16:08:46.565 :  INFO : [u'NATIVE_APP']
20170510 16:08:46.565 :  INFO : ${Contexts} = [u'NATIVE_APP']
20170510 16:08:46.565 :  INFO : [u'NATIVE_APP']
Ending test:   RobotFrameworkTest1.TestSuite5.TestCase012
```

从执行结果看，我们只获取到了一个 Context，这是因为当前 APP 中就只打开了一个 Context。

4.2.10　Switch To Context

Switch To Context 关键字用来在多个 Context 之间进行切换，接收[context_name]一个参数。

我们在手机端经常会调用到 H5 页面，并且会在页面中做切换，Switch To Context 就可以帮助我们完成页面的切换操作。通过上面讲到的 Get Contexts 关键字可以获取到所有的 Context，然后执行 Switch To Context 关键字就可以直接做切换了。

【示例】切换到 APP 的 NATIVE_APP 这个 Context 下面。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
${Contexts} Get Contexts
log ${Contexts}
Switch To Context NATIVE_APP
```

4.2.11　Get Elements

Get Elements 关键字用来获取通过某个 locator 匹配到的所有的元素，接收 [locator | first_element_only=False | fail_on_error=True]三个参数，其中，当 first_element_only=True 时，只会返回匹配到的第一个元素；当 fail_on_error=True 时，如果没有获取到任何元素，那么该关键字会执行失败。

【示例】使用 Get Elements 关键字和 locator 为 class=android.widget.Button 来获取匹配到的所有的元素，在这里依旧采用上面使用的 APP 来做操作示例。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
${Elements} Get Elements    class=android.widget.Button
log ${Elements}
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase013
20170510 16:18:17.221 :  INFO : msg:find class=android.widget.Button
20170510 16:18:18.995 :  INFO : ${Elements} =
[<appium.webdriver.webelement.WebElement
(session="205dfa7e-1289-40de-a555-34880166ccfa", element="1")>,
<appium.webdriver.webelement.WebElement
(session="205dfa7e-1289-40de-a555-34880166ccfa", elemen...
20170510 16:18:18.995 :  INFO : [<appium.webdriver.webelement.WebElement
(session="205dfa7e-1289-40de-a555-34880166ccfa", element="1")>,
<appium.webdriver.webelement.WebElement
(session="205dfa7e-1289-40de-a555-34880166ccfa", element="2")>]
Ending test:  RobotFrameworkTest1.TestSuite5.TestCase013
```

从执行结果看，根据 class=android.widget.Button 获取到了两个元素。

4.2.12　Get Element Attribute

Get Element Attribute 关键字用来获取某个元素的属性值，接收[locator | attribute]两个参数。

【示例】使用 Get Element Attribute 关键字和 locator 为 class=android.widget.Button 来获取 name 属性的值，在这里依旧采用上面使用的 APP 来做操作示例。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
${Attribute}    Get Element Attribute    class=android.widget.Button name

log ${Attribute}
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase014
20170510 16:43:35.268 :  INFO : msg:find class=android.widget.Button
20170510 16:43:36.897 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="fcbb5ef0-d97c-4127-8719-0e6ef14c2403", element="1")>,
<appium.webdriver.webelement.WebElement
(session="fcbb5ef0-d97c-4127-8719-0e6ef14c2403", element="2")>]
20170510 16:43:36.897 :  INFO : CAUTION: 'class=android.widget.Button' matched
2 elements - using the first element only
20170510 16:43:36.949 :  INFO : Element 'class=android.widget.Button' attribute
'name' value '计算'
20170510 16:43:36.950 :  INFO : ${Attribute} = 计算
20170510 16:43:36.953 :  INFO : 计算
Ending test:   RobotFrameworkTest1.TestSuite5.TestCase014
```

从执行结果看，获取到的 name 属性值为“计算”，而且根据执行日志可知使用 class=android.widget.Button 匹配到了两个元素，当匹配到了多个元素时默认只会使用第一个元素。

4.2.13　Get Network Connection Status 和 Set Network Connection Status

Get Network Connection Status 关键字用来获取手机的网络连接状态。在获取到连接状态后，会返回不同的数字。

Set Network Connection Status 关键字用来设置手机的网络连接状态，如表 4-2-1 所示。

表 4-2-1　手机网络连接的状态码

Status 状态码	数据流量连接	WiFi 连接	飞行模式	说明
0	0	0	0	不打开任何连接
1	0	0	1	打开飞行模式
2	0	1	0	只打开 wifi 网络
4	1	0	0	只打开数据连接
6	1	1	0	打开所有的网络连接

【示例】

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
${Attribute}    Get Network Connection Status
log ${Attribute}
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase015
20170513 15:16:05.063 :  INFO : ${Attribute} = 0
20170513 15:16:05.063 :  INFO : 0
Ending test:  RobotFrameworkTest1.TestSuite5.TestCase015
```

从执行结果看，获取到连接状态码为 0，说明没有打开任何网络连接。

4.2.14 Element Attribute Should Match

Element Attribute Should Match 关键字用来判断元素的属性值是否和预期值匹配，接收 [locator | attr_name | match_pattern | regexp=False]四个参数。当通过 locator 识别到元素超过 1 个元素时，会默认选择第一个元素；attr_name 参数代表所选元素的属性的名称；match_pattern 参数代表预期匹配值；regexp 代表了匹配时是否通过正则表达式来进行匹配。

【示例 1】使用 Element Attribute Should Match 关键字来判断通过 locator 为 id=com.example.calculator:id/factorone 和 attr_name=name 获取到的值是否能和预期值匹配。在这里依旧采用上面使用的 APP 来做操作示例。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Element Attribute Should Match
    id=com.example.calculator:id/factorone name    输入框
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase017
20170513 15:48:54.838 :  INFO : msg:find id=com.example.calculator:id/factorone
20170513 15:48:54.839 :  INFO : prefix: id
20170513 15:48:54.840 :  INFO : criteria: com.example.calculator:id/factorone
20170513 15:48:56.622 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="6708d455-65b8-4af7-8d7c-ea86e7f039f3", element="1")>]
20170513 15:48:56.745 :  INFO : Element 'id=com.example.calculator:id/factorone'
attribute 'name' is '输入框'
Ending test:  RobotFrameworkTest1.TestSuite5.TestCase017
```

从执行结果看，“Element 'id=com.example.calculator:id/factorone' attribute 'name' is '输入框'”，正好和“输入框”能匹配上。

【示例 2】改用 locator 为 class=android.widget.Button 和 attr_name=name 来进行匹配验证，并且看一下当 locator 匹配到多个元素时是如何进行处理的。

```
Open Application    http://localhost:4723/wd/hub    platformName=Android
    platformVersion=22  deviceName=98YFBP522VSU
    app=C:/Users/yongqing/Desktop/app-debug.apk
    appPackage=com.example.calculator  appActivity=MainActivity
Element Attribute Should Match class=android.widget.Button name    计*  True
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite5.TestCase018
20170520 15:06:29.527 :  INFO : msg:find class=android.widget.Button
20170520 15:06:29.527 :  INFO : prefix: class
20170520 15:06:29.527 :  INFO : criteria: android.widget.Button
20170520 15:06:31.214 :  INFO : elements:
[<appium.webdriver.webelement.WebElement
(session="dce6f097-98d2-4632-8708-598e2f693721", element="1")>,
<appium.webdriver.webelement.WebElement
(session="dce6f097-98d2-4632-8708-598e2f693721", element="2")>]
20170520 15:06:31.214 :  INFO : CAUTION: 'class=android.widget.Button' matched
2 elements - using the first element only
20170520 15:06:31.243 :  INFO : Element 'class=android.widget.Button' attribute
'name' is '计*'
Ending test:  RobotFrameworkTest1.TestSuite5.TestCase018
```

从执行结果看，当通过 class=android.widget.Button 获取到两个元素时，默认只会使用第一个元素，能和预期结果匹配上。

4.2.15　Element Name Should Be 和 Element Value Should Be

这是两个断言关键字，对元素 Element 的 name 和 value 的值进行直接断言处理。

Element Name Should Be 关键字用来断言指定元素的名称是否和预期的一致，接收[locator | expected]两个参数。

Element Value Should Be 关键字用来断言指定元素的 value 值是否和预期的一致，接收[locator | expected]两个参数。

4.2.16　AppiumLibrary 库其他的常见自动化关键字

表 4-2-2 中描述了 AppiumLibrary 库中剩余其他关键字的用法。

表 4-2-2　AppiumLibrary 库中其他关键字的用法

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Close Application</td><td>关闭掉当前已经打开的 APP Application，该关键字不需要接收任何参数，但是使用该关键字的前提是已经打开了一个 APP Application</td></tr>
<tr><td>Close All Applications</td><td>关闭掉当前已经打开的所有 APP Application，该关键字不需要接收任何参数</td></tr>
<tr><td>Background App</td><td>让当前 APP Application 运行在后台，该关键字接收一个参数[后台运行的时间]，示例：<table><tr><td>Background App</td><td>5s</td></tr></table></td></tr>
<tr><td>Capture Page Screenshot</td><td>获取当前页面的截图，如果对该关键字没有传入任何参数，就会默认将获取到的截图命名为 appium-screenshot-<counter>.png，并且保存到 RobotFramework 的运行日志目录下，在案例运行失败时，经常需要保存截图来辅助定位和分析</td></tr>
<tr><td>Set Appium Timeout</td><td>设置 Appium 的超时时间，该关键字接收一个参数[超时的时长，单位为秒]</td></tr>
<tr><td>Get Appium Timeout</td><td>获取不同关键字使用的超时时间</td></tr>
<tr><td>Go Back</td><td>返回到浏览器的上一个操作步骤，相当于浏览器的后退按钮功能</td></tr>
<tr><td>Go To Url</td><td>在默认的浏览器上打开一个 url 地址，该关键字接收一个参数[待打开的 url]</td></tr>
<tr><td>Hide Keyboard</td><td>隐藏当前操作设备的键盘。需要注意的是，该关键字在安卓手机上执行时可以不加任何参数，在 iOS 手机上执行时可以使用` key_name `按特定的键</td></tr>
<tr><td>Input Password</td><td>该关键字和 Input Text 关键字的功能基本一致，只是当使用该关键字时，日志中不会记录输入的 password</td></tr>
<tr><td>Input Value</td><td>该关键字只适用于 iOS 设备，可以接收两个参数[locator | text]</td></tr>
<tr><td>Lock</td><td>锁定手机设备</td></tr>
<tr><td>Long Press</td><td>该关键字用于模拟长按 APP 界面的某个元素，接收一个参数[locator]</td></tr>
<tr><td>Long Press Keycode</td><td>该关键字用于模拟长按手机设备上的一个按键，接收两个参数[keycode | metastate=None]。注意，该关键字只适用于安卓设备</td></tr>
<tr><td>Press Keycode</td><td>该关键字只适用于安卓设备，模拟对键盘上的按键进行操作，如模拟 Ctrl 或者 Alt 键。该关键字接收两个参数[keycode | metastate=None]</td></tr>
<tr><td>Pull File</td><td>该关键字用于从手机设备上下拉文件，接收两个参数[path | decode=False]。其中，path 参数指的是设备上文件的路径；decode 参数指的是是否按照 base64 来解码，默认为 False</td></tr>
<tr><td>Pull Folder</td><td>该关键字和 Pull File 关键字用法相似，接收两个参数[path | decode=False]。其中，path 参数指的是设备上文件的路径；decode 参数指的是是否按照 base64 来解码，默认为 False</td></tr>
<tr><td>Push File</td><td>该关键字的功能和 Pull File 关键字相反，用于向手机设备上传文件。该关键字接收三个参数[path | data | encode=False]：path 参数指的是设备的路径；data 参数指的是待传入到设备上的文件；encode 参数指的是是否按照 base64 来编码，默认为 False</td></tr>
<tr><td>Remove Application</td><td>该关键字用于移除设备上的 application，接收一个参数[application_id]，示例：<table><tr><td>Remove Application</td><td>com.example.calculator</td></tr></table></td></tr>
<tr><td>Register Keyword To Run On Failure</td><td>该关键字用于指出在案例执行失败时需要执行哪个关键字。示例：<table><tr><td>Register Keyword To Run On Failure</td><td>Log Source</td></tr></table></td></tr>
</table>

（续表）

关键字	使用描述
Scroll	该关键字模拟屏幕滚动或者滑动，接收两个参数[start_locator | end_locator]，可以模拟从一个元素滑动到另一个元素
Scroll To	该关键字和 Scroll 关键字很类似，但是只接收一个参数[locator]
Shake	该关键字用于模拟摇动手机设备
Swipe	模拟手机滑动，从一个点滑动到另一个点。该关键字接收五个参数[start_x | start_y | end_x | end_y | duration=1000]，滑动时，点的定位采用坐标的形式
Tap	该关键字模拟对屏幕元素进行单击，接收一个参数[locator]
Zoom	该关键字用于缩放一个元素，接收三个参数[locator | percent=200% | steps=1]：第一个参数为定位元素的方式；第二个参数为缩放的百分比，如果不传入，就默认放大两倍：第三个参数为元素缩放的步骤数（number of steps in the zoom action），默认为 1

第 5 章 Web自动化测试

5.1 Selenium Web 自动化

Selenium 出来已经有很多年了，从最初的 Selenium1 到后来的 Selenium2，也变得越来越成熟，而且已经被很多公司广泛使用。Selenium 在发展的过程中分了很多模块，这里我们主要介绍 WebDriver。WebDriver 已经被很多浏览器所兼容。WebDriver 在自动化脚本和浏览器之间充当的角色和之前介绍的 Appium 很像。

由于现在很多的浏览器都已经主动支持和兼容了 WebDriver，所以 WebDriver 在启动后会确认浏览器的 native component 是否存在可用而且版本匹配，接着在目标浏览器里启动使用 Selenium 自己设计定义的协议（WebDriver Wire Protocol）。WebDriver Wire 协议是通用的，也就是说不管是 FirefoxDriver 还是 ChromeDriver 等，启动之后都会在某一个端口启动基于这套协议的 Web 服务。WebDriver Wire 协议是一套基于 RESTful 的 Web 服务。在调用 WebDriver 的时候，实际上是给在浏览器上启动的 RESTful 服务监听端口发送 HTTP 请求，以 WebDriver Wire 协议规定的 JSON 格式的字符串来告诉 Selenium 希望浏览器执行什么样的操作。

5.1.1 Selenium 和 Robot Framework Selenium2Library 库介绍

通过网址 http://www.seleniumhq.org/ 可以访问 Selenium 官网。从该网站上可以下载到各种浏览器运行需要的 Driver，如图 5-1-1 和图 5-1-2 所示。

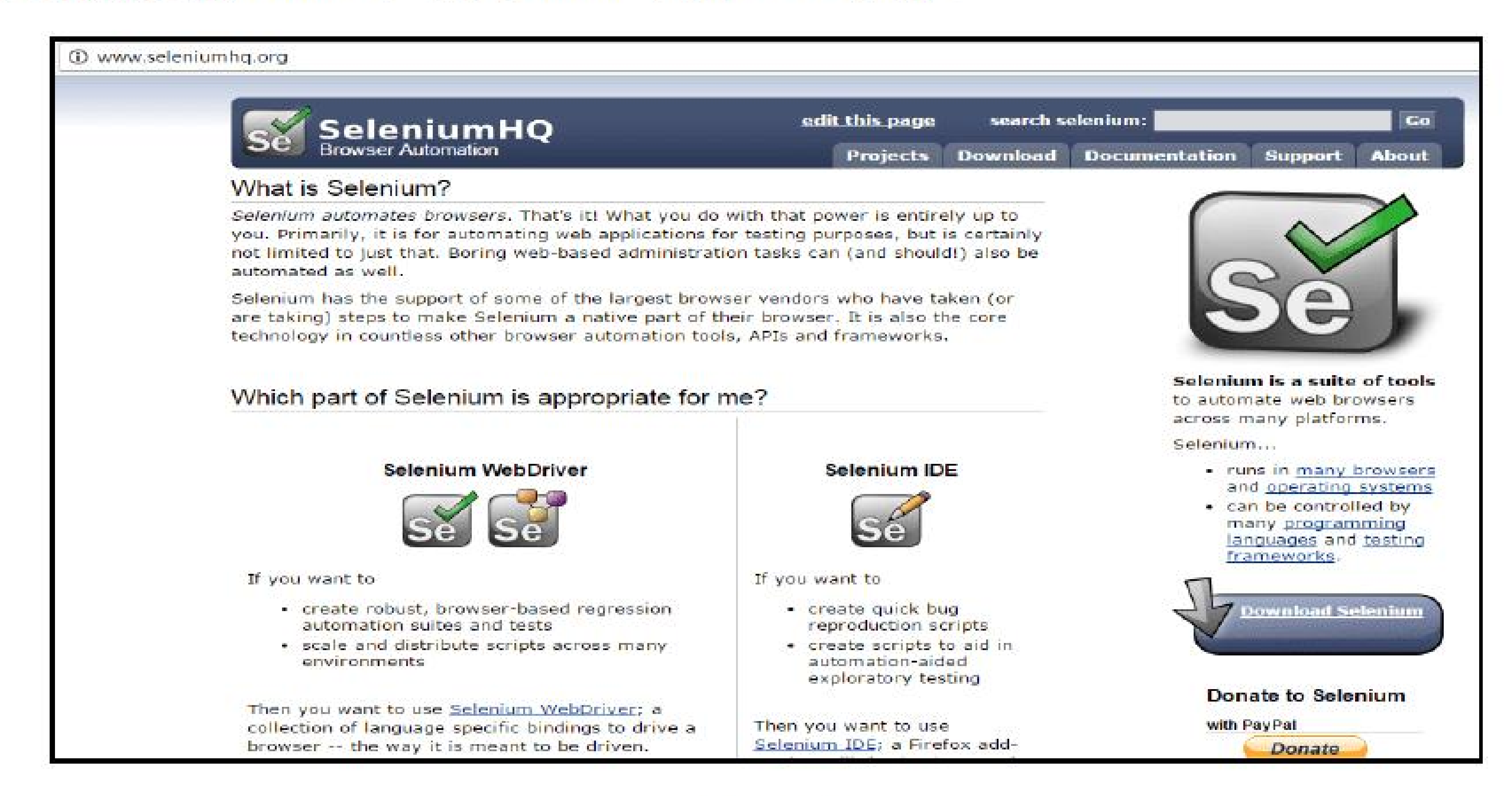


图 5-1-1

Third Party Drivers, Bindings, and Plugins

Selenium can be extended through the use of plugins. Here are a number of plugins created and maintained by third parties. For more information on how to create your own plugin or have it listed, consult the docs.

Please note that these plugins are not supported, maintained, hosted, or endorsed by the Selenium project. In addition, be advised that the plugins listed below are not necessarily licensed under the Apache License v.2.0. Some of the plugins are available under another free and open source software license; others are only available under a proprietary license. Any questions about plugins and their license of distribution need to be raised with their respective developer(s).

Third Party Browser Drivers NOT DEVELOPED by seleniumhq

Browser					
Mozilla GeckoDriver	0.16.1	change log	issue tracker	Implementation Status	Released 2017-04-26
Google Chrome Driver	2.29	change log	issue tracker	selenium wiki page	Released 2017-04-04
Opera	2.27		issue tracker	selenium wiki page	Released 2017-04-04
Microsoft Edge Driver			issue tracker	Implementation Status	
GhostDriver	(PhantomJS)		issue tracker	SeConf talk	
HtmlUnitDriver	2.26		issue tracker		Released 2017-04-04
SafariDriver			issue tracker		
Windows Phone			issue tracker		
Windows Phone	4.14.028.10		issue tracker		Released 2013-11-23
Selendroid - Selenium for Android			issue tracker		

图 5-1-2

从 http://www.seleniumhq.org/docs/ 地址可以查询到关于 Selenium 的文档和介绍等信息，如图 5-1-3 所示。

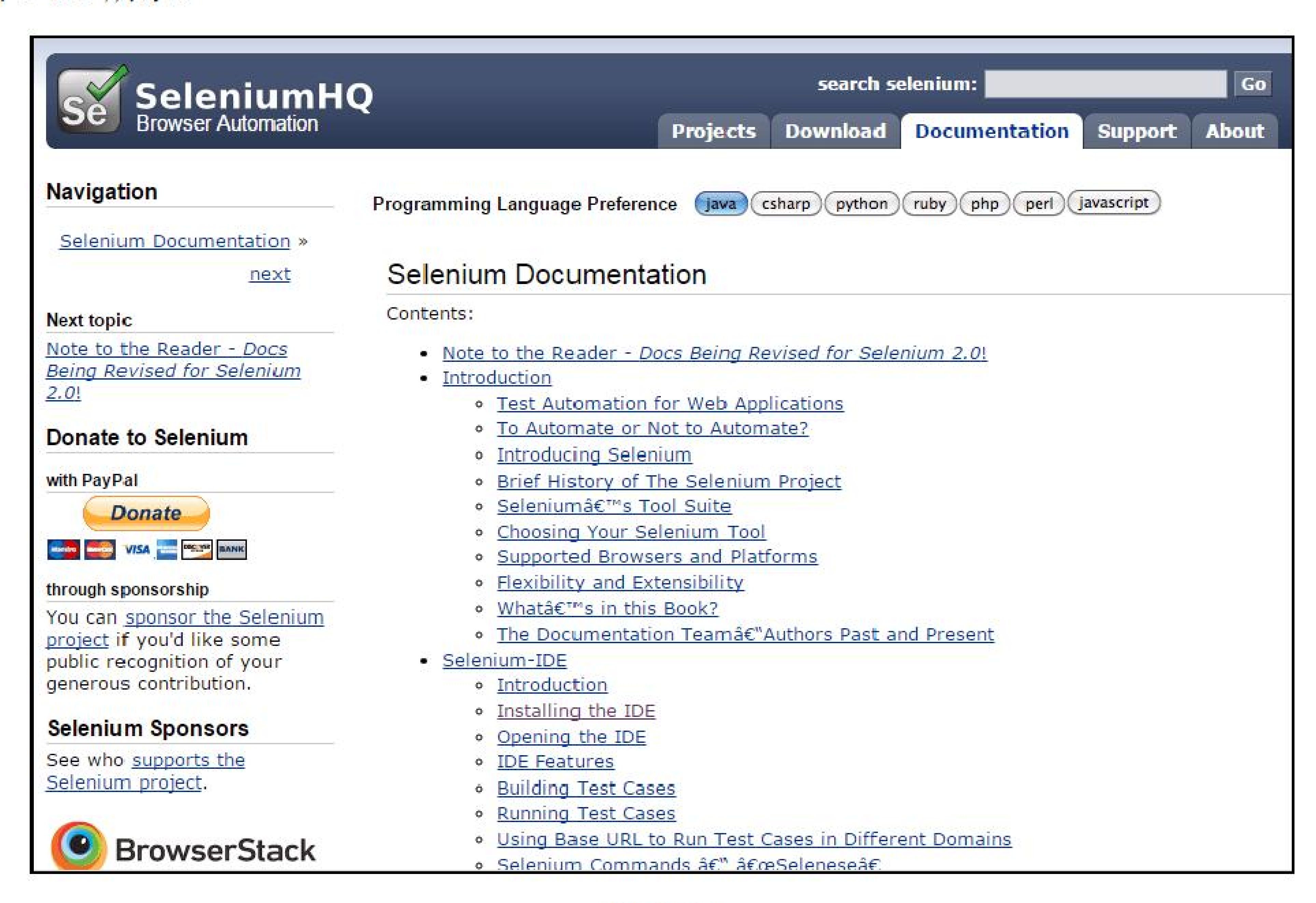


图 5-1-3

5.1.2 Open Browser 和 Close Browser

在 Selenium2Library 库中，Open Browser 关键字用来打开一个指定的浏览器，该关键字接收如表 5-1-1 所示的参数。

表 5-1-1 Open Browser 关键字接收的参数

参数（Arguments）	说明
url	在浏览器中需要打开的 url 地址
Browser	指定需要打开的浏览器类型，包括 IE、Firefox、Chrome、Opera、Safari 等常用的浏览器，默认使用 Firefox
Alias	设定的浏览器实例的别名，可以用于浏览器之间的切换，默认为 None
remote_url	是否启用通过 remote server 的形式来访问，默认为 False
desired_capabilities	可以指定的配置参数，默认为 None
ff_profile_dir	该参数主要针对火狐浏览器，可以通过该参数指定 Firefox profile 路径，默认为 None

Close Browser 关键字用来关闭一个已经打开的当前浏览器。

【示例 1】打开谷歌浏览器，url 地址中输入 http://www.baidu.com。

```
Open Browser    http://www.baidu.com    chrome
```

执行结果如图 5-1-4 所示。

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0002
20170529 15:22:55.284 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0002
```



图 5-1-4

【示例 2】打开 IE 浏览器，url 地址中输入 http://www.baidu.com，然后关闭浏览器。

```
Open Browser    http://www.baidu.com    ie
Close Browser
```

执行结果如图 5-1-5 所示。

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0001
20170529 15:31:01.937 :  INFO : Opening browser 'ie' to base url
'http://www.baidu.com'
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0001
```



图 5-1-5

另外在使用 IE 浏览器时，需要使浏览器中安全标签下的每个区域是否启用保护模式保持一致，要么全部启用，要么全部不启用，如图 5-1-6 所示。不然的话容易出现类似“WebDriverException: Message: Unexpected error launching Internet Explorer. Protected Mode settings are not the same for all zones. Enable Protected Mode must be set to the same value (enabled or disabled) for all zones.”的报错。



图 5-1-6

5.1.3 Input Text

Input Text 关键字用于模拟向一个输入框中输入文字内容。该关键字接收两个参数[locator | text]：locator 参数指的是定位界面元素的方式，text 参数指的是需要输入的内容。

【示例】打开百度页面，向输入框中输入“Robot FrameWork”，这里采用 id 的方式来定位界面的输入框元素，可以采用谷歌浏览器自带的开发者工具查看该输入框的元素，如图 5-1-7 所示。



图 5-1-7

```
Open Browser    http://www.baidu.com    chrome
Input Text  id=kw    Robot FrameWork
```

执行结果如图 5-1-8 所示。

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0002
20170529 15:44:23.918 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20170529 15:44:32.068 :  INFO : Typing text 'Robot FrameWork' into text field
'id=kw'
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0002
```

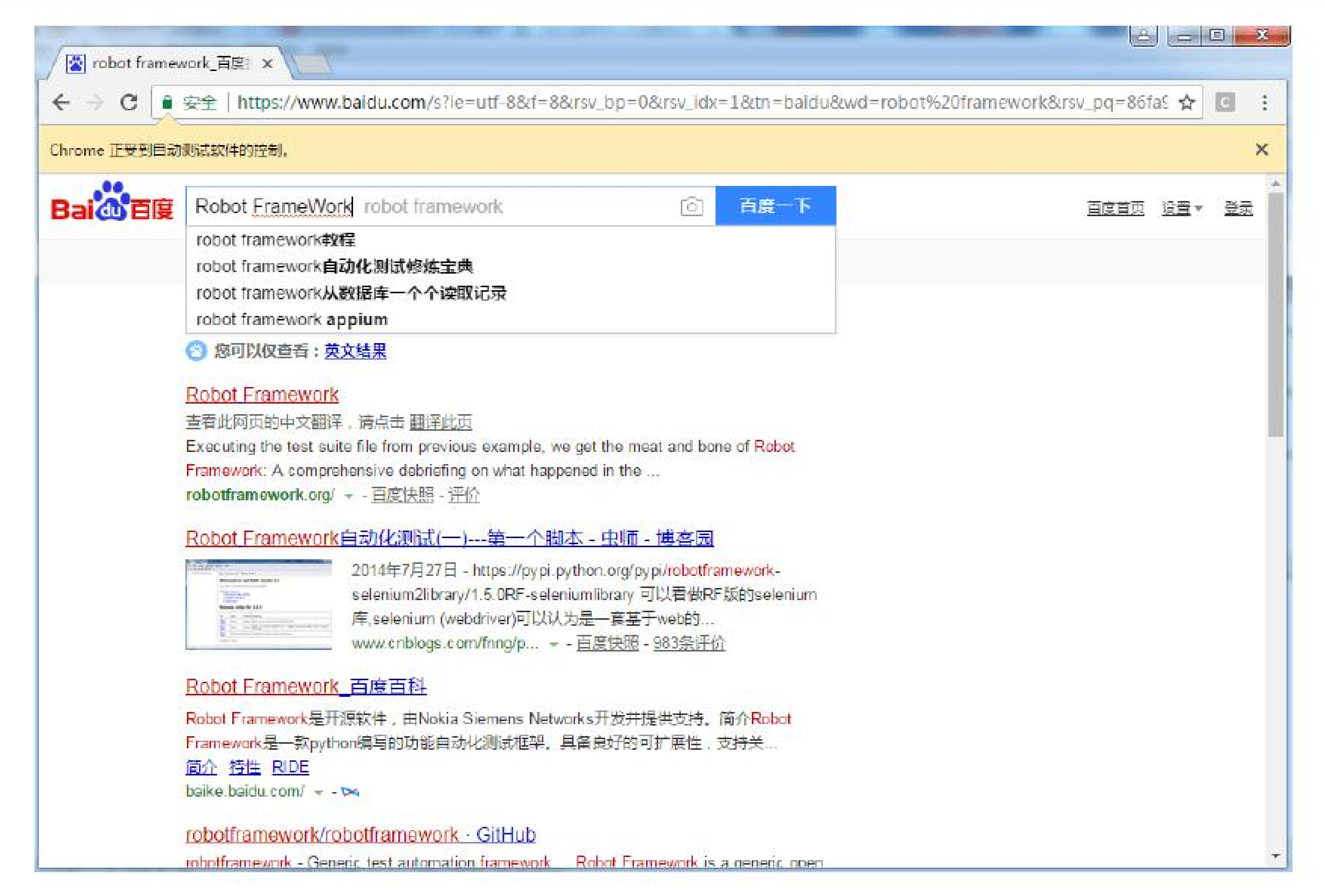


图 5-1-8

5.1.4 Click Button

Click Button 关键字用于模拟单击页面中的按钮，接收一个参数[locator]。

【示例】打开百度页面，向输入框中输入“Robot FrameWork”后，单击“百度一下”按钮，进行搜索。

这里通过 id=su 来定位“百度一下”这个按钮，如图 5-1-9 所示。

```
Open Browser    http://www.baidu.com    chrome
Input Text  id=kw    Robot FrameWork
Click Button    id=su
```

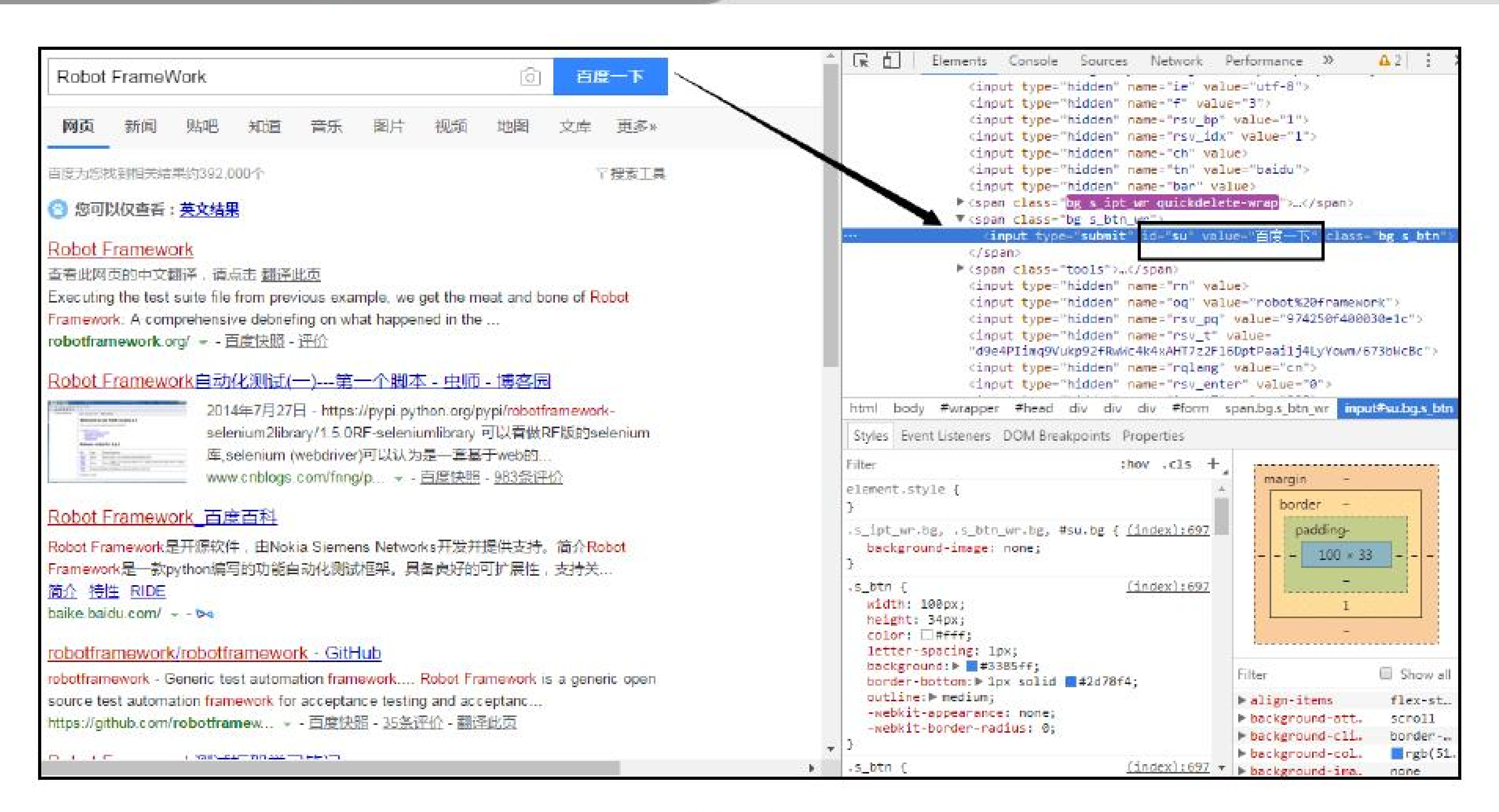


图 5-1-9

执行结果如图 5-1-10 所示。

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0002
20170529 16:01:02.161 : INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20170529 16:01:12.505 : INFO : Typing text 'Robot FrameWork' into text field
'id=kw'
20170529 16:01:12.917 : INFO : Clicking button 'id=su'.
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0002
```



图 5-1-10

5.1.5　Click Element

Click Element 关键字用于模拟单击一个通过 locator 定位到的具体元素，可以是通过 id 或者 name、xpath 等。该关键字接收一个参数[locator]。

【示例】这里我们继续访问百度的首页，通过 id 来定位一个元素，并且使用 Click Element 关键字来模拟单击这个元素定位到的按钮，如图 5-1-11 所示。



图 5-1-11

```
Open Browser    http://www.baidu.com    chrome
Click Element   id=su
Sleep    2
Close Browser
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0003
20180728 09:59:02.582 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20180728 09:59:14.272 :  INFO : Clicking element 'id=su'.
20180728 09:59:17.158 :  INFO : Slept 2 seconds
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0003
```

5.1.6　Click Link

Click Link 关键字用于模拟单击一个链接。该关键字接收一个参数[locator]。

【示例 1】这里我们继续访问百度的首页，通过模拟单击百度首页右上角的“地图”链接来说明此关键字的使用。下面通过 href="http://map.baidu.com" 来模拟单击这个链接，如图 5-1-12 所示。



图 5-1-12

```
Open Browser    http://www.baidu.com    chrome
Click Link  http://map.baidu.com
Sleep    2
Close Browser
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0004
20180728 10:10:37.328 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20180728 10:10:47.171 :  INFO : Clicking link 'http://map.baidu.com'.
20180728 10:10:53.170 :  INFO : Slept 2 seconds
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0004
```

【示例 2】Click Link 关键字除了上面说到的直接通过 href 链接来定位元素外，也可以通过 id 或者 name 来定位，这里以 name 为例进行演示。

```
Open Browser    http://www.baidu.com
Click Link  name=tj_trmap
Sleep    5
Close Browser
```

执行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0005
20180728 10:22:35.467 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
```

```
20180728 10:22:44.921 :  INFO : Clicking link 'name=tj_trmap'.
20180728 10:22:57.497 :  INFO : Slept 5 seconds
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0005
```

运行时可以看到正在加载百度地图的界面，如图 5-1-13 所示。

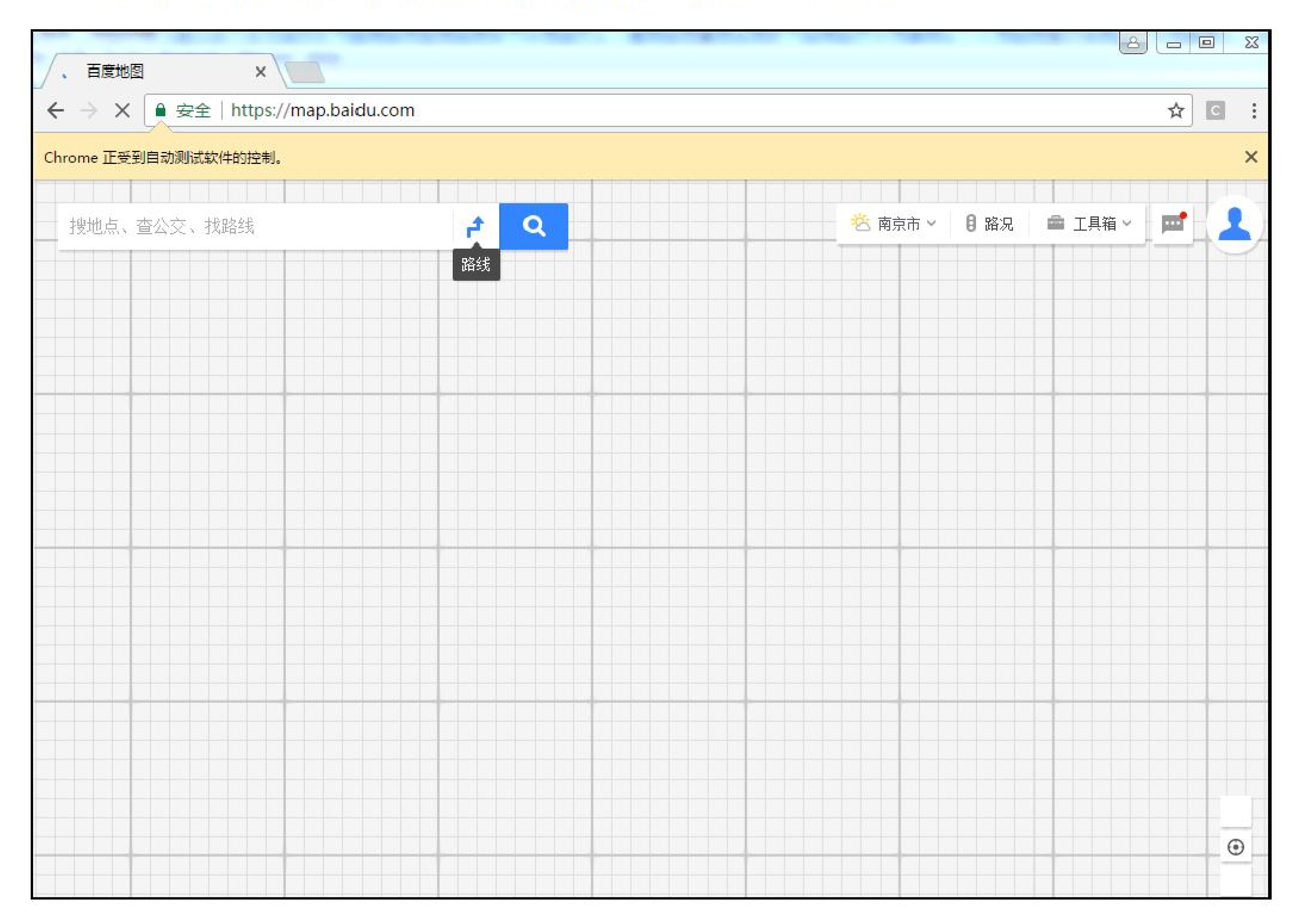


图 5-1-13

5.1.7　Add Cookie、Get Cookie 和 Delete Cookie

Add Cookie 关键字用于模拟向本地浏览器中添加一个 Cookie 缓存，Cookie 也是我们在做 Web 自动化测试时经常需要用到的一个概念。该关键字接收[name | value | path=None | domain=None | secure=None | expiry=None]这几个参数，如表 5-1-2 所示。

表 5-1-2　Add Cookie 关键字的参数

参数	说明
name	Cookie 的名称
Value	Cookie 的具体值
path	Cookie 对应的路径，如果不填就默认为 None
domain	Cookie 对应的域名，如果不填就默认为 None
secure	Cookie 的安全属性，用来保证 Cookie 安全的。如果一个 Cookie 被设置了 Secure=true，那么这个 Cookie 只能用 HTTPS 协议发送给服务器，用 HTTP 协议是不发送的
expiry	Cookie 的过期时间，如果不填，默认为空

这里以登录到作者的博客园后浏览器中显示的 Cookie 为示例，如图 5-1-14 所示，上面表格中对应的字段都可以在浏览器的 Cookie 中看到。

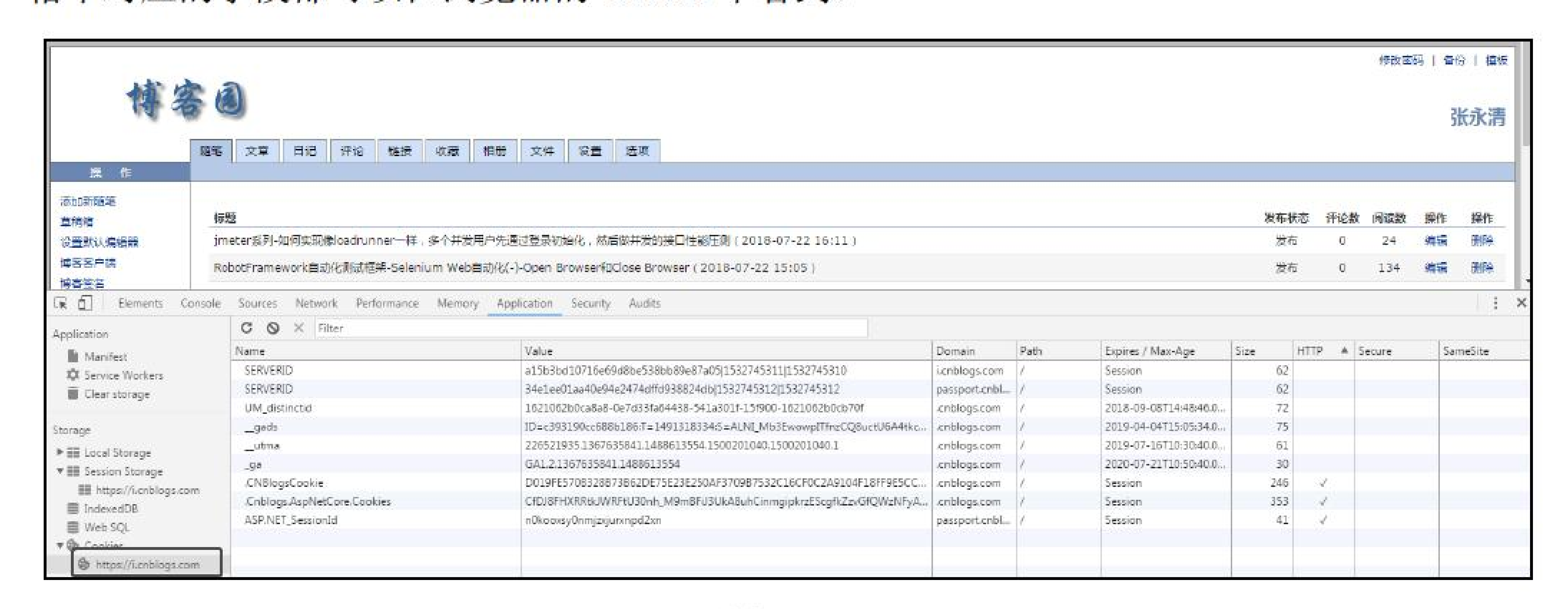


图 5-1-14

Get Cookie 关键字用于获取浏览器中缓存的所有 Cookie，这个关键字后面不需要加任何的参数。

Delete Cookie 关键字用于删除浏览器中缓存的 Cookie。该关键字接收一个参数[name]，用于标志需要删除的 Cookie 的名称。

【示例】这里我们以访问百度首页为例，添加一个名称为 book 的 Cookie，并且将该 Cookie 的值写为 Robot Framework、secure 属性设置为 True，然后通过 Get Cookies 关键字来获取所有的 Cookie，再删除我们自己添加的 Cookie。

```
Open Browser    http://www.baidu.com    chrome
Add Cookie  book    Robot Framework /    baidu.com    true
${cookie}  Get Cookies
log ${cookie}
Sleep   50
Delete Cookie   book
${cookienew}    Get Cookies
log ${cookienew}
Sleep   2
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0006
20180728 11:02:14.076 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20180728 11:02:24.488 :  INFO : ${cookie} =
H_PS_PSSID=26523_1443_26433_21112_26924_20930;
BAIDUID=53A9ECC2223045BB4D28064D8CCC2428:FG=1; PSTM=1532746930;
BIDUPSID=53A9ECC2223045BB4D28064D8CCC2428; delPer=0; BD_HOME=0;
BD_UPN=12314353; book= Robot Framework
20180728 11:02:24.489 :  INFO : H_PS_PSSID=26523_1443_26433_21112_26924_20930;
BAIDUID=53A9ECC2223045BB4D28064D8CCC2428:FG=1; PSTM=1532746930;
```

```
BIDUPSID=53A9ECC2223045BB4D28064D8CCC2428; delPer=0; BD_HOME=0;
BD_UPN=12314353; book=Robot Framework
20180728 11:03:14.490 :  INFO : Slept 50 seconds
20180728 11:03:15.772 :  INFO : ${cookienew} =
H_PS_PSSID=26523_1443_26433_21112_26924_20930;
BAIDUID=53A9ECC2223045BB4D28064D8CCC2428:FG=1; PSTM=1532746930;
BIDUPSID=53A9ECC2223045BB4D28064D8CCC2428;delPer=0;BD_HOME=0;BD_UPN=12314353
20180728 11:03:15.773 :  INFO : H_PS_PSSID=26523_1443_26433_21112_26924_20930;
BAIDUID=53A9ECC2223045BB4D28064D8CCC2428:FG=1; PSTM=1532746930;
BIDUPSID=53A9ECC2223045BB4D28064D8CCC2428;delPer=0;BD_HOME=0;BD_UPN=12314353
20180728 11:03:17.775 :  INFO : Slept 2 seconds
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0006
```

从运行的日志结果看，通过“Add Cookie book Robot Framework/baidu.comtrue”，可以添加一个 Cookie，并且可以将该 Cookie 的 secure 属性设置为 True。在运行的过程中，通过浏览器自带的开发者工具，可以看到名称叫 book 的 Cookie 已经成功添加完成，如图 5-1-15 所示。

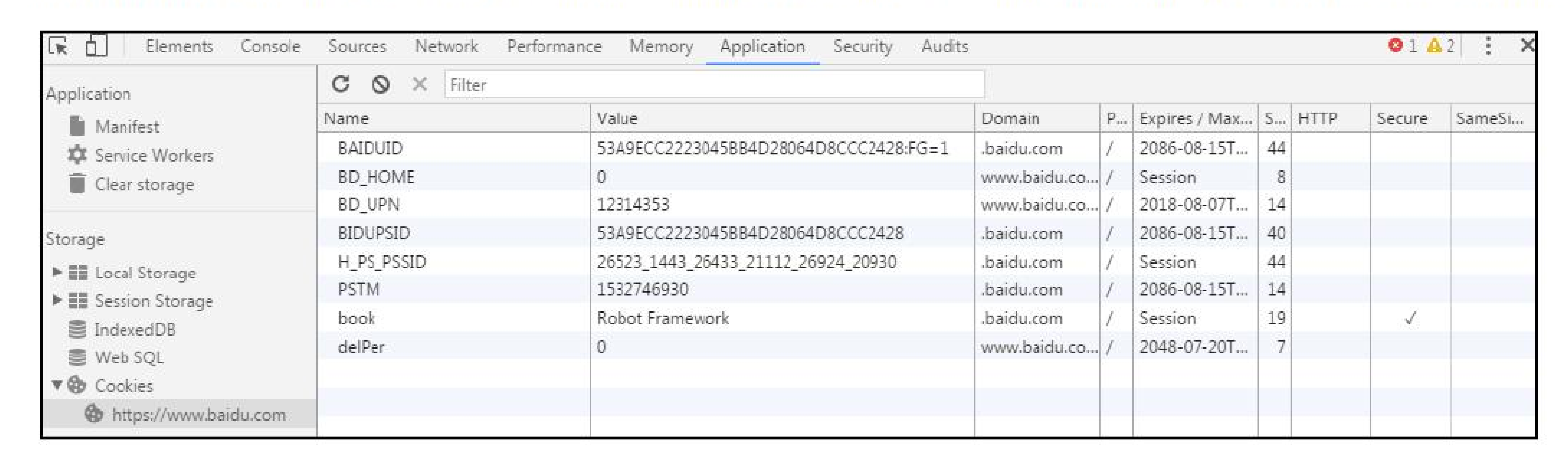


图 5-1-15

在使用 Get Cookies 关键字时，可以获取到浏览器打开后里面所有的 Cookie，除了获取到这里我们自己添加的 Cookie 外，还获取到了访问百度首页时百度首页在本地浏览器中保存的 Cookie。最后通过 Delete Cookie 关键字删除名称为 book 的 Cookie 后，再通过 Get Cookies 关键字来获取所有的 Cookie 就已经不能获取到名称为 book 的 Cookie 信息了。

5.1.8　Get All Links

Get All Links 关键字用来获取所有页面上所有的 href 链接的元素对应的 id，链接对应的元素中没有 id 时，就以一个空字符串代替。

【示例】我们继续访问百度，找出百度首页中存在多少个链接数。

```
Open Browser    http://www.baidu.com/   chrome
${links}    Get All Links
log ${links}
Sleep    2
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0007
20180728 11:31:18.136 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com/'
20180728 11:31:28.558 :  INFO : get_attribute id
20180728 11:31:28.566 :  INFO : get_attribute id
20180728 11:31:28.574 :  INFO : get_attribute id
20180728 11:31:28.582 :  INFO : get_attribute id
20180728 11:31:28.592 :  INFO : get_attribute id
20180728 11:31:28.599 :  INFO : get_attribute id
20180728 11:31:28.607 :  INFO : get_attribute id
20180728 11:31:28.617 :  INFO : get_attribute id
20180728 11:31:28.624 :  INFO : get_attribute id
20180728 11:31:28.633 :  INFO : get_attribute id
20180728 11:31:28.640 :  INFO : get_attribute id
20180728 11:31:28.648 :  INFO : get_attribute id
20180728 11:31:28.657 :  INFO : get_attribute id
20180728 11:31:28.664 :  INFO : get_attribute id
20180728 11:31:28.673 :  INFO : get_attribute id
20180728 11:31:28.682 :  INFO : get_attribute id
20180728 11:31:28.690 :  INFO : get_attribute id
20180728 11:31:28.699 :  INFO : get_attribute id
20180728 11:31:28.706 :  INFO : get_attribute id
20180728 11:31:28.716 :  INFO : get_attribute id
20180728 11:31:28.723 :  INFO : get_attribute id
20180728 11:31:28.732 :  INFO : get_attribute id
20180728 11:31:28.740 :  INFO : get_attribute id
20180728 11:31:28.747 :  INFO : get_attribute id
20180728 11:31:28.756 :  INFO : get_attribute id
20180728 11:31:28.764 :  INFO : get_attribute id
20180728 11:31:28.775 :  INFO : get_attribute id
20180728 11:31:28.782 :  INFO : get_attribute id
20180728 11:31:28.790 :  INFO : get_attribute id
20180728 11:31:28.798 :  INFO : get_attribute id
20180728 11:31:28.806 :  INFO : get_attribute id
20180728 11:31:28.814 :  INFO : get_attribute id
20180728 11:31:28.822 :  INFO : get_attribute id
20180728 11:31:28.831 :  INFO : ${links} = [u'result_logo', u'quickdelete', u'',
u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'',
u'', u'', u'', u'', u'', u'', u'', u'setf', u'', u'', u'', u'', u'', u'jgwab']
20180728 11:31:28.833 :  INFO : [u'result_logo', u'quickdelete', u'', u'', u'',
u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'', u'',
u'', u'', u'', u'', u'', u'setf', u'', u'', u'', u'', u'', u'jgwab']
20180728 11:31:30.833 :  INFO : Slept 2 seconds
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0007
```

5.1.9 Choose File

做自动化测试时，我们经常遇到的就是上传文件了。这里我们可以使用 Choose File 关键字完成该操作。Choose File 关键字接收[locator | file_path]两个关键字，如表 5-1-3 所示。

表 5-1-3　Choose File 关键字接收的关键字

关键字	说明
locator	上传文件时，上传文件对应的输入框的输入字段可以通过我们常见的元素定位的方式来定位到
file_path	这个参数指的是需要上传的文件的本地路径

【示例】这里我们还是以百度首页为例。百度首页支持图片搜索，很多经常使用百度引擎进行搜索的朋友可能都用过，它有点类似通过输入一张图片，然后查找和这张图片的相似图片功能一样。在目前人工智能技术大热的时代，这个功能被越来越多的电商用于商品搜索。

我们首先打开百度首页，然后通过上面介绍的 Click Element 关键字模拟单击输入框旁边的📷按钮来切换到图片上传模式。切换完后，通过 Choose File 关键字来上传文件，使用 xpath 来定位（locator）。之后就是选择磁盘中的一张个人图片上传。

在获取 xpath 路径时，我们可以通过 Chrome 浏览器自带的开发者工具来直接复制到 xpath 路径，如图 5-1-16 所示。



图 5-1-16

```
Open Browser    http://www.baidu.com/   chrome
Click Element   //*[@id="form"]/span[1]/span
Sleep   10
Choose File //*[@id="form"]/div/div[2]/div[2]/input    E:\\zhangyongqing.bmp
Sleep   10
Close Browser
```

运行结果如下（这里的运行结果是从自动化执行 report 中来查看的）：

SUITE RobotFrameworkTest1

```
javascript:expandAll('s1')javascript:collapseAll('s1')C:\Users\yongqing\AppData\Local\Temp\RIDExpjcyt.d\log.html - s1#s1
```

Full Name:	RobotFrameworkTest1
Source:	F:\project\RobotFrameworkTest1\RobotFrameworkTest1
Start / End / Elapsed:	20180728 11:52:36.817 / 20180728 11:53:08.127 / 00:00:31.310
Status:	1 critical test, 1 passed, 0 failed 1 test total, 1 passed, 0 failed

00:00:31.284SUITE TestSuite6

```
javascript:expandAll('s1-s1')javascript:collapseAll('s1-s1')C:\Users\yongqing
\AppData\Local\Temp\RIDExpjcyt.d\log.html - s1-s1#s1-s1
```

Full Name:	RobotFrameworkTest1.TestSuite6
Source:	F:\project\RobotFrameworkTest1\RobotFrameworkTest1\TestSuite6.txt
Start / End / Elapsed:	20180728 11:52:36.841 / 20180728 11:53:08.125 / 00:00:31.284
Status:	1 critical test, 1 passed, 0 failed 1 test total, 1 passed, 0 failed

00:00:31.108TEST TestCase0008

javascript:expandAll('s1-s1-t1')javascript:collapseAll('s1-s1-t1')C:\Users\yongqing\AppData\Local\Temp\RIDExpjcyt.d\log.html - s1-s1-t1#s1-s1-t1

Full Name:	RobotFrameworkTest1.TestSuite6.TestCase0008
Start / End / Elapsed:	20180728 11:52:37.016 / 20180728 11:53:08.124 / 00:00:31.108
Status:	**PASS** (critical)

00:00:09.825KEYWORD Selenium2Library . Open Browser http://www.baidu.com/, chrome

```
javascript:expandAll('s1-s1-t1-k1')javascript:collapseAll('s1-s1-t1-k1')C:\Us
ers\yongqing\AppData\Local\Temp\RIDExpjcyt.d\log.html -
s1-s1-t1-k1#s1-s1-t1-k1
```

Documentation:	Opens a new browser instance to given URL.
Start / End / Elapsed:	20180728 11:52:37.017 / 20180728 11:52:46.842 / 00:00:09.825

11:52:37.017	INFO	Opening browser 'chrome' to base url 'http://www.baidu.com/'	

00:00:00.119KEYWORD Selenium2Library . Click Element //*[@id="form"]/span[1]/span

```
javascript:expandAll('s1-s1-t1-k2')javascript:collapseAll('s1-s1-t1-k2')C:\Us
ers\yongqing\AppData\Local\Temp\RIDExpjcyt.d\log.html -
s1-s1-t1-k2#s1-s1-t1-k2
```

Documentation:	Click element identified by `locator`.

Start / End / Elapsed:	20180728 11:52:46.842 / 20180728 11:52:46.961 / 00:00:00.119

11:52:46.843	INFO	Clicking element '//*[@id="form"]/span[1]/span'.

00:00:10.001KEYWORD BuiltIn . Sleep 10

```
javascript:expandAll('s1-s1-t1-k3')javascript:collapseAll('s1-s1-t1-k3')C:\Us
ers\yongqing\AppData\Local\Temp\RIDExpjcyt.d\log.html -
s1-s1-t1-k3#s1-s1-t1-k3
```

Documentation:	Pauses the test executed for the given time.
Start / End / Elapsed:	20180728 11:52:46.962 / 20180728 11:52:56.963 / 00:00:10.001

11:52:56.962	INFO	Slept 10 seconds

00:00:00.095KEYWORD Selenium2Library . Choose File //*[@id="form"]/div/div[2]/div[2]/input, E:\\zhangyongqing.bmp

```
javascript:expandAll('s1-s1-t1-k4')javascript:collapseAll('s1-s1-t1-k4')C:\Us
ers\yongqing\AppData\Local\Temp\RIDExpjcyt.d\log.html -
s1-s1-t1-k4#s1-s1-t1-k4
```

Documentation:	Inputs the `file_path` into file input field found by `locator`.
Start / End / Elapsed:	20180728 11:52:56.964 / 20180728 11:52:57.059 / 00:00:00.095

00:00:10.002KEYWORD BuiltIn . Sleep 10

```
javascript:expandAll('s1-s1-t1-k5')javascript:collapseAll('s1-s1-t1-k5')C:\Us
ers\yongqing\AppData\Local\Temp\RIDExpjcyt.d\log.html -
s1-s1-t1-k5#s1-s1-t1-k5
```

Documentation:	Pauses the test executed for the given time.
Start / End / Elapsed:	20180728 11:52:57.060 / 20180728 11:53:07.062 / 00:00:10.002

11:53:07.061	INFO	Slept 10 seconds

00:00:01.061KEYWORD Selenium2Library . Close Browser

```
javascript:expandAll('s1-s1-t1-k6')javascript:collapseAll('s1-s1-t1-k6')C:\Us
ers\yongqing\AppData\Local\Temp\RIDExpjcyt.d\log.html -
s1-s1-t1-k6#s1-s1-t1-k6
```

Documentation:	Closes the current browser.
Start / End / Elapsed:	20180728 11:53:07.063 / 20180728 11:53:08.124 / 00:00:01.061

为了能看到整个过程，我们捕获了运行过程中关键步骤执行时的实际效果图片。从图片中

可以清楚地看到图片从上传到上传后百度引擎自动执行搜索的一个过程，如图 5-1-17、图 5-1-18、图 5-1-19 所示。



图 5-1-17



图 5-1-18



图 5-1-19

5.1.10 Get Text

Get Text 关键字用来获取文本内容，该关键字接收[locator]这一个关键字。locator 可以通过 id、name、xpath 等来定位。

【示例 1】这里我们依然以百度首页为例，获取 name="tj_trnews" 包含的文本内容，如图 5-1-20 所示。

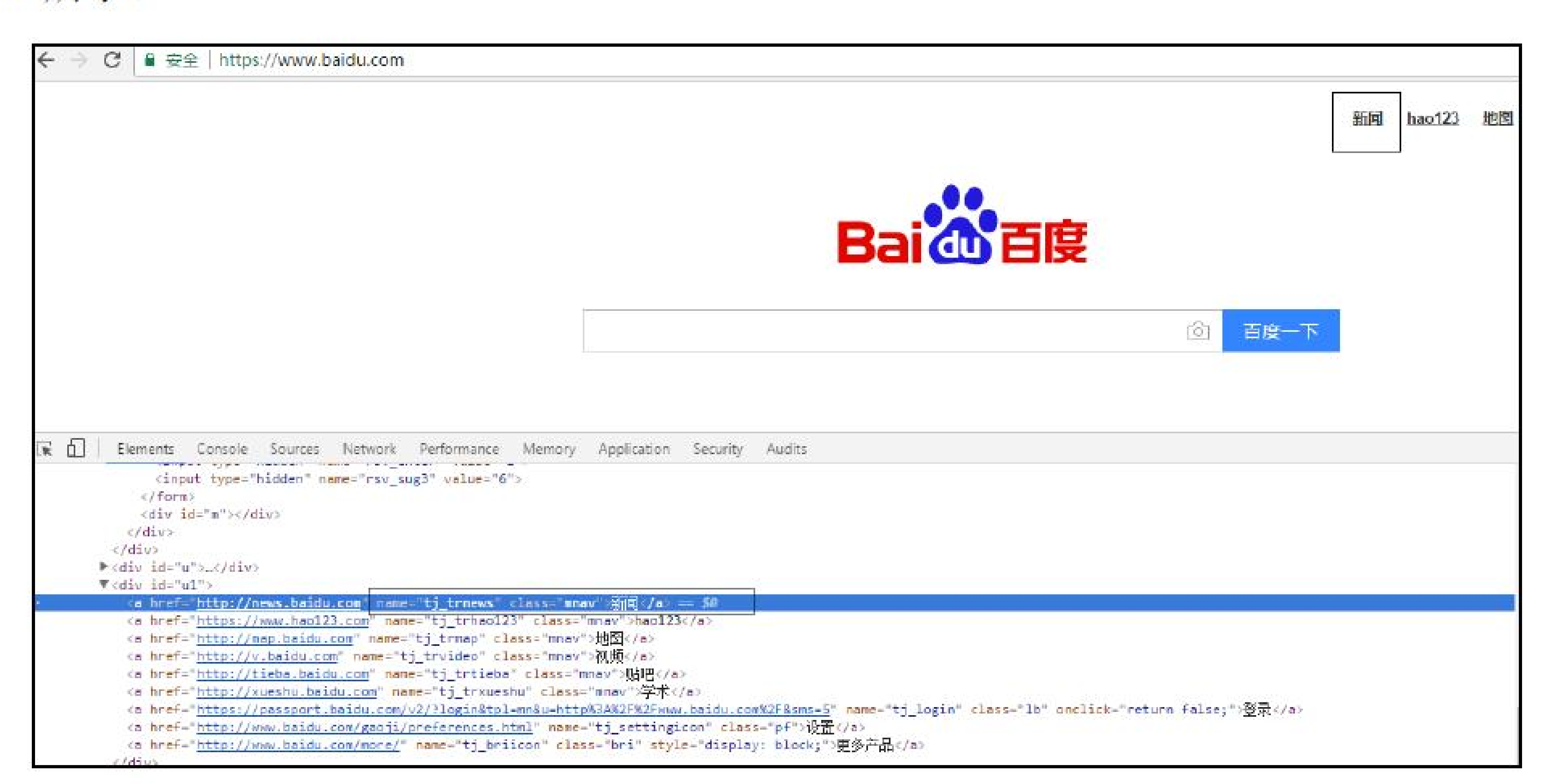


图 5-1-20

```
Open Browser    http://www.baidu.com/   chrome
${text} Get Text    name=tj_trnews
log ${text}
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0009
20180728 13:04:33.505 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com/'
20180728 13:04:42.409 :  INFO : ${text} = 新闻
20180728 13:04:42.411 :  INFO : 新闻
```

```
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0009
```

【示例 2】在上面的示例中，我们的 locator 是通过 name 来定位的，本示例换成 xpath 来试试。

```
Open Browser    http://www.baidu.com/   chrome
${text} Get Text     //*[@id="u1"]/a[1]
log ${text}
Close Browser
```

运行结果（从自动化执行 report 中来查看）如图 5-1-21 所示。

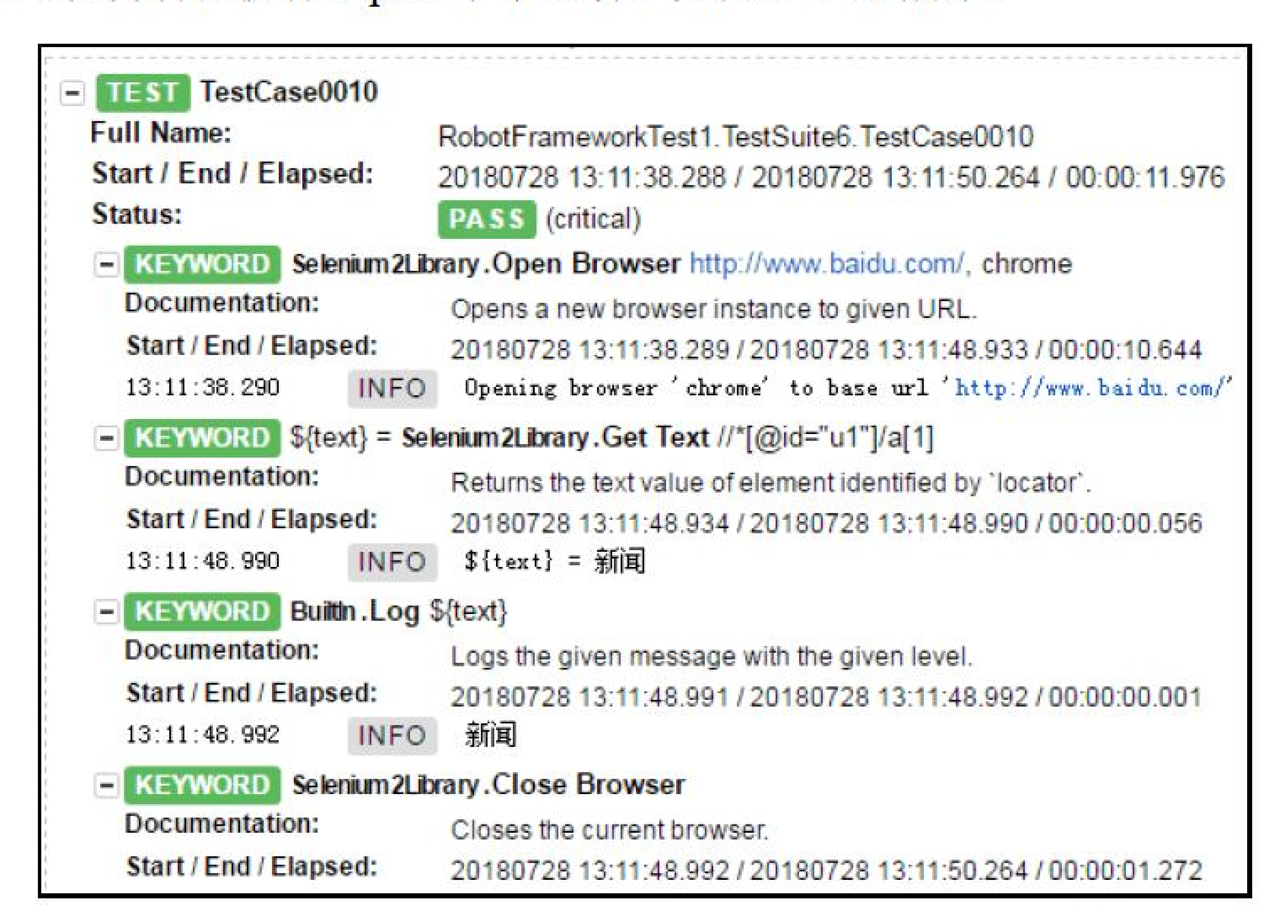


图 5-1-21

我们发现使用 xpath 的效果是一样的。

5.1.11 Get Title

Get Title 关键字用来获取浏览器网页的 title，该关键字后面不需要接收任何参数。

【示例】这里我们模拟访问百度首页，然后获取百度首页的 Title

```
Open Browser    http://www.baidu.com/   chrome
${title}    Get Title
log ${title}
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0011
20180728 13:18:33.227 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com/'
20180728 13:18:42.213 :  INFO : ${title} = 百度一下，你就知道
20180728 13:18:42.214 :  INFO : 百度一下，你就知道
```

```
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0011
```

5.1.12　Get Value

Get Value 关键字用于获取某个元素标签对应的 value 属性，该关键字接收[locator]这一个参数。locator 可以通过 id、name、xpath 等进行定位。

【示例】这里我们以访问博客园的登录页面为示例，获取登录按钮对应的 value，如图 5-1-22 所示。

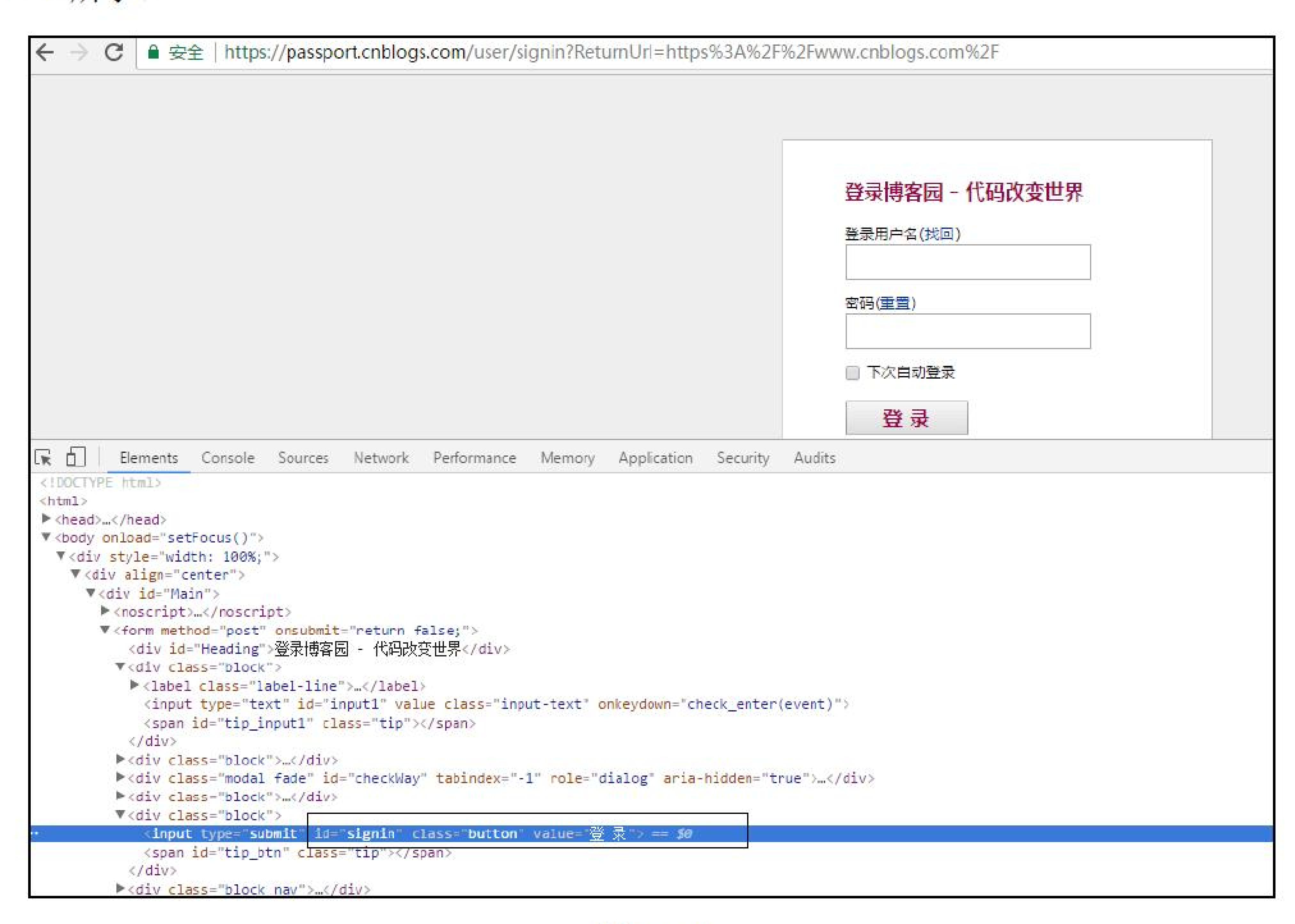


图 5-1-22

```
Open Browser
    https://passport.cnblogs.com/user/signin?ReturnUrl=https%3A%2F%2Fwww.cnbl
ogs.com%2F  chrome
${value}    Get Value   id=signin
log ${value}
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0012
20180728 13:34:51.578 :  INFO : Opening browser 'chrome' to base url
'https://passport.cnblogs.com/user/signin?ReturnUrl=https%3A%2F%2Fwww.cnblogs
.com%2F'
20180728 13:35:00.116 :  INFO : get_attribute value
20180728 13:35:00.124 :  INFO : ${value} = 登 录
20180728 13:35:00.125 :  INFO : 登 录
```

```
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0012
```

5.1.13 Get Webelements 和 Get Webelement

Get Webelements 关键字用来获取所有获得的 WebElement 对象的列表，接收[locator]一个参数。locator 可以通过 id、name、xpath 等进行定位。

Get Webelement 关键字和 Get Webelements 类似，只不过 Get Webelement 只会返回匹配到的第一个 WebElement 对象。

【示例 1】访问百度首页，然后根据 locator 为 name=tj_trnews 来获取可以匹配到的所有 WebElement 对象的列表。

```
Open Browser    http://www.baidu.com    chrome
${ement}    Get Webelements name=tj_trnews
log ${ement}
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0013
20180728 14:54:42.820 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20180728 14:54:50.182 :  INFO : ${ement} =
[<selenium.webdriver.remote.webelement.WebElement
(session="47f749d3fcd2d5037a56e6ada80f38ba",
element="0.8260127734608302-1")>]
20180728 14:54:50.183 :  INFO :
[<selenium.webdriver.remote.webelement.WebElement
(session="47f749d3fcd2d5037a56e6ada80f38ba",
element="0.8260127734608302-1")>]
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0013
```

【示例 2】访问百度首页，然后根据 locator 为 name=tj_trnews 来获取匹配到的第一个 WebElement 对象。

```
Open Browser    http://www.baidu.com    chrome
${ement}    Get Webelement  name=tj_trnews
log ${ement}
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0014
20180728 15:01:58.469 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20180728 15:02:07.093 :  INFO : ${ement} =
<selenium.webdriver.remote.webelement.WebElement
(session="0382b83b32515ea731a4497ab3699131",
element="0.20523497043976824-1")>
20180728 15:02:07.094 :  INFO : <selenium.webdriver.remote.webelement.WebElement
```

```
(session="0382b83b32515ea731a4497ab3699131",
element="0.20523497043976824-1")>
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0014
```

5.1.14　Get Window Titles

Get Window Titles 用来获取当前已经打开的浏览器窗口的所有 Title。该关键字不需要接收任何参数。

【示例】这里我们打开百度的首页，然后单击首页右上角的“新闻”链接按钮，跳转到百度新闻页面，最后获取该窗口下的所有 Title。

```
Open Browser    http://www.baidu.com    chrome
Click Link  新闻
${title1}   Get Window Titles
log ${title1}
Close All Browsers
```

运行结果如图 5-1-23 所示。

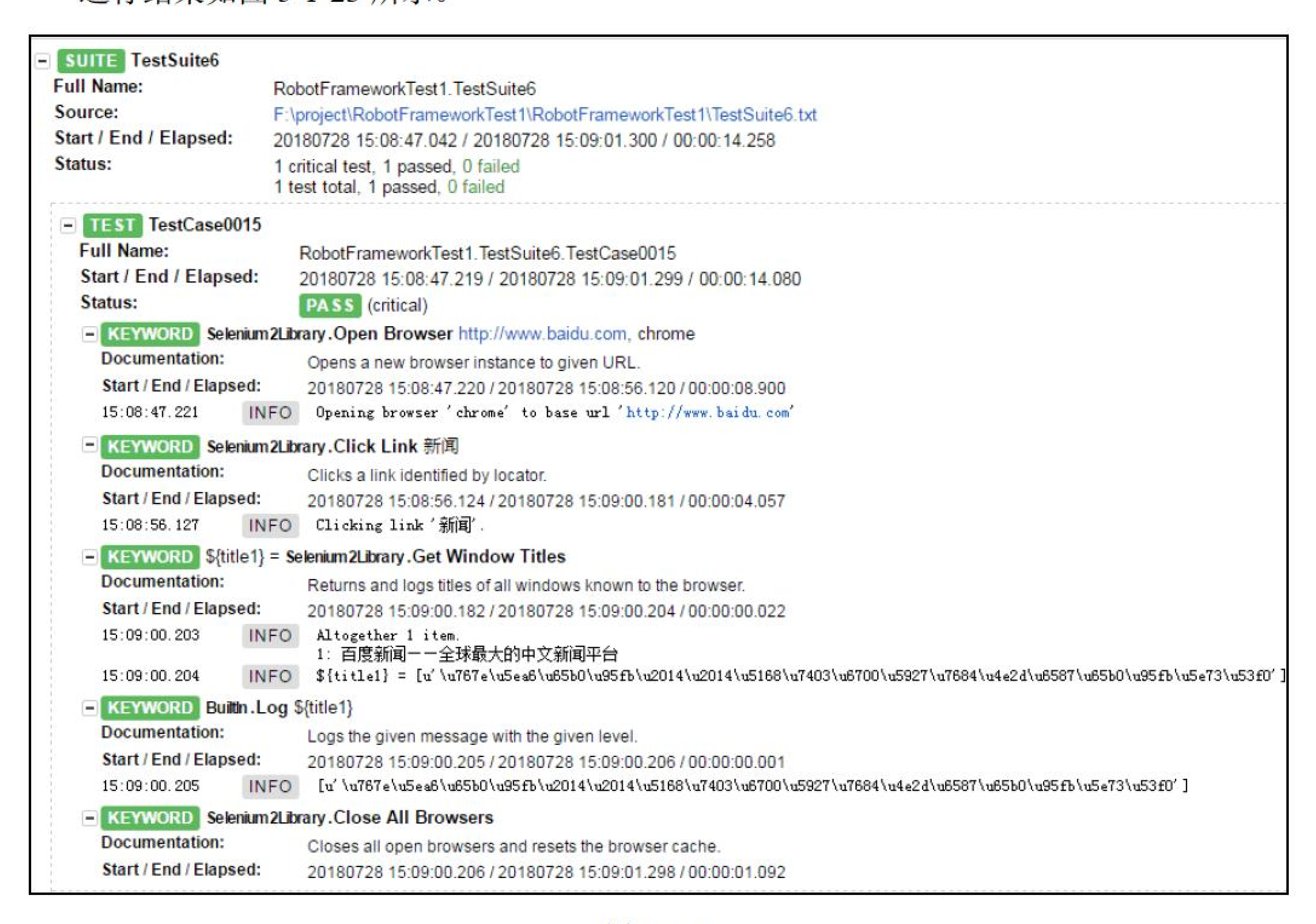


图 5-1-23

5.1.15　Go Back 和 Go To

Go Back 和 Go To 关键字分别用来模拟对浏览器进行后退和前进的操作。Go Back 关键

字不需要接收任何参数。Go To 关键字接收[url]一个参数，url 是一个需要跳转到的地址。

【示例】这里我们首先访问百度首页，之后单击“新闻”链接，跳转到百度新闻页面，然后执行 Go Back 回退到百度首页，最后使用 Go To 跳转到博客园首页，在每一个操作中，我们都记录了浏览器的窗口标题。

```
Open Browser    http://www.baidu.com    chrome
Click Link  新闻
${title1}   Get Window Titles
log ${title1}
Go Back
${title2}   Get Window Titles
log ${title2}
Go To   https://www.cnblogs.com/
${title3}   Get Window Titles
log ${title3}
Close All Browsers
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0016
20180728 15:26:17.443 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20180728 15:26:24.906 :  INFO : Clicking link '新闻'.
20180728 15:26:27.947 :  INFO :
Altogether 1 item.
```

1: 百度新闻——全球最大的中文新闻平台

```
20180728 15:26:27.948 :  INFO : ${title1} =
[u'\u767e\u5ea6\u65b0\u95fb\u2014\u2014\u5168\u7403\u6700\u5927\u7684\u4e2d\u
6587\u65b0\u95fb\u5e73\u53f0']
20180728 15:26:27.949 :  INFO :
[u'\u767e\u5ea6\u65b0\u95fb\u2014\u2014\u5168\u7403\u6700\u5927\u7684\u4e2d\u
6587\u65b0\u95fb\u5e73\u53f0']
20180728 15:26:28.239 :  INFO :
Altogether 1 item.
```

1: 百度一下，你就知道

```
20180728 15:26:28.240 :  INFO : ${title2} =
[u'\u767e\u5ea6\u4e00\u4e0b\uff0c\u4f60\u5c31\u77e5\u9053']
20180728 15:26:28.242 :  INFO :
[u'\u767e\u5ea6\u4e00\u4e0b\uff0c\u4f60\u5c31\u77e5\u9053']
20180728 15:26:28.244 :  INFO : Opening url 'https://www.cnblogs.com/'
20180728 15:26:31.021 :  INFO :
Altogether 1 item.
```

1: 博客园 - 开发者的网上家园

```
20180728 15:26:31.021 :  INFO : ${title3} = [u'\u535a\u5ba2\u56ed -
\u5f00\u53d1\u8005\u7684\u7f51\u4e0a\u5bb6\u56ed']
```

```
20180728 15:26:31.022 : INFO : [u'\u535a\u5ba2\u56ed -
\u5f00\u53d1\u8005\u7684\u7f51\u4e0a\u5bb6\u56ed']
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0016
```

从运行的日志可以很清楚地看到 Go Back 和 Go To 关键字在使用时起到的效果。

5.1.16 Get List Items

Get List Items 关键字用来获取页面中一个下拉列表中的所有下拉选项。该关键字接收[locator]一个参数，locator 可以通过 id 或者 name 等来进行元素定位。

【示例】我们以访问百度贴吧中的一个下拉列表框为例来对该关键字的使用进行说明，如图 5-1-24 所示。

图 5-1-24

```
Open Browser    http://tieba.baidu.com/f/search/adv?red_tag=u3387165643
    chrome
@{Items}    Get List Items   name=sm
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0023
20180728 23:27:16.635 : INFO : Opening browser 'chrome' to base url
'http://tieba.baidu.com/f/search/adv?red_tag=u3387165643'
20180728 23:27:24.714 : INFO : get_attribute multiple
20180728 23:27:24.878 : INFO : @{Items} = [ 按时间倒序 | 按时间顺序 | 按相关性排序 ]
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0023
```

5.1.17 Get Selected List Value

Get Selected List Value 关键字用于获取页面中选中的一个下拉列表的 Value 值。关键字接收[locator]一个参数，locator 可以通过 id 或者 name 来进行元素定位。

【示例】这里我们依然以访问百度贴吧中的一个下拉列表框为例来对该关键字的使用进行说明，如图 5-1-25 所示。

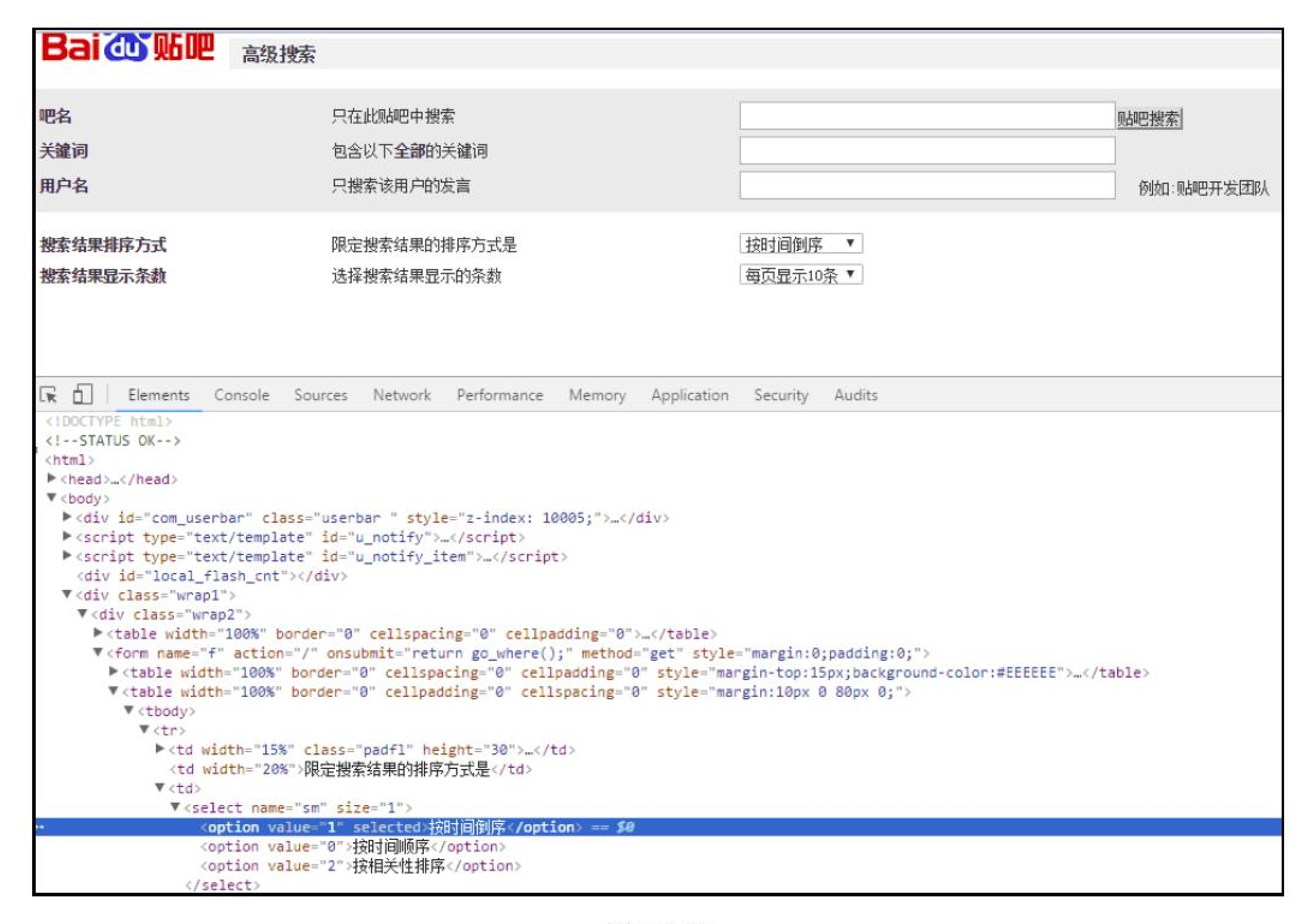


图 5-1-25

```
Open Browser    http://tieba.baidu.com/f/search/adv?red_tag=u3387165643
    chrome
${Value}    Get Selected List Value    name=sm
log ${Value}
Close Browser
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0025
20180728 23:54:56.379 :  INFO : Opening browser 'chrome' to base url
'http://tieba.baidu.com/f/search/adv?red_tag=u3387165643'
20180728 23:55:05.545 :  INFO : get_attribute multiple
20180728 23:55:05.599 :  INFO : get_attribute value
20180728 23:55:05.609 :  INFO : ${Value} = 1
20180728 23:55:05.610 :  INFO : 1
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0025
```

从运行结果看，刚好与我们通过浏览器的开发工具看到的 value 值完全一致，如图 5-1-26 所示。

```
▼<tbody>
  ▼<tr>
    ▶<td width="15%" class="padfl" height="30">…</td>
     <td width="20%">限定搜索结果的排序方式是</td>
    ▼<td>
      ▼<select name="sm" size="1">
          <option value="1" selected>按时间倒序</option> == $0
          <option value="0">按时间顺序</option>
          <option value="2">按相关性排序</option>
        </select>
      </td>
```

图 5-1-26

5.1.18　Select From List

Select From List 关键字用来模拟从指定的下拉列表中选择指定的下拉列表选项。该关键字接收[locator | *items]多个参数，locator 可以通过 id、name 来进行元素的定位。当列表中传入多个值时，默认选择最后一条，如果传入的是一个空列表，就会默认选择这个列表中的所有值。

【示例】这里我们继续访问百度贴吧，目标是通过 Select From List 关键字选中 按相关性排序 ▼ 这个选项。我们通过浏览器的开发者工具可以看到 按相关性排序 ▼ 这个选项对应的 value 为 2，所以我们在关键字的参数中传入的参数为 2，如图 5-1-27 所示。

```
Open Browser    http://tieba.baidu.com/f/search/adv?red_tag=u3387165643
    chrome
Select From List    name=sm     2
sleep   5
Close Browser
```

```
tieba.baidu.com/f/search/adv?red_tag=u3387165643
Baidu贴吧  高级搜索
吧名            只在此贴吧中搜索            [          ] 贴吧搜索
关键词          包含以下全部的关键词        [          ]
用户名          只搜索该用户的发言          [          ]  例如:贴吧开发团队
搜索结果排序方式  限定搜索结果的排序方式是    按时间倒序 ▼
搜索结果显示条数  选择搜索结果显示的条数      每页显示10条 ▼
Elements  Console  Sources  Network  Performance  Memory  Application  Security  Audits
<!DOCTYPE html>
<!--STATUS OK-->
<html>
▶<head>…</head>
▼<body>
  ▶<div id="com_userbar" class="userbar " style="z-index: 10005;">…</div>
  ▶<script type="text/template" id="u_notify">…</script>
  ▶<script type="text/template" id="u_notify_item">…</script>
   <div id="local_flash_cnt"></div>
  ▼<div class="wrap1">
    ▼<div class="wrap2">
      ▶<table width="100%" border="0" cellspacing="0" cellpadding="0">…</table>
      ▼<form name="f" action="/" onsubmit="return go_where();" method="get" style="margin:0;padding:0;">
        ▶<table width="100%" border="0" cellspacing="0" cellpadding="0" style="margin-top:15px;background-color:#EEEEEE">…</table>
        ▼<table width="100%" border="0" cellpadding="0" cellspacing="0" style="margin:10px 0 80px 0;">
          ▼<tbody>
            ▼<tr>
              ▶<td width="15%" class="padfl" height="30">…</td>
               <td width="20%">限定搜索结果的排序方式是</td>
              ▼<td>
                ▼<select name="sm" size="1">
                    <option value="1" selected>按时间倒序</option>
                    <option value="0">按时间顺序</option>
                    <option value="2">按相关性排序</option> == $0
```

图 5-1-27

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0031
20180729 09:53:40.279 :  INFO : Opening browser 'chrome' to base url
'http://tieba.baidu.com/f/search/adv?red_tag=u3387165643'
20180729 09:53:49.543 :  INFO : Selecting option(s) '2' from list 'name=sm'.
20180729 09:53:49.582 :  INFO : get_attribute multiple
20180729 09:53:54.675 :  INFO : Slept 5 seconds
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0031
```

从实际执行的效果看，已经可以成功选中 按相关性排序 ▼ 这个列表，如图 5-1-28 所示。



图 5-1-28

5.1.19 Selenium2Library 库其他的自动化测试关键字介绍

表 5-1-4 中介绍了 Selenium2Library 库中剩余的其他关键字的用法。

表 5-1-4 Selenium2Library 库中其他关键字的用法

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Page Should Contain</td><td>该关键字用来判断当前窗口页面是否包含了指定的文本内容，接收[text | loglevel=INFO]两个参数，如果不传入日志级别时，默认 loglevel=INFO。示例：
<table><tr><td>Page Should Contain</td><td>Robot Framework</td></tr></table></td></tr>
<tr><td>Page Should Contain Button</td><td>该关键字用来判断当前窗口页面是否包含了指定的 Button 按钮，接收[locator | message= | loglevel=INFO]三个参数，和 Page Should Contain 关键字的使用方式很类似，locator 参数可以通过 id、name、value 等来进行元素定位。示例：
<table>
<tr><td>Open Browser</td><td>http://www.baidu.com</td><td>chrome</td></tr>
<tr><td>Page Should Contain Button</td><td>id=su</td><td>包含“百度一下”按钮</td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table></td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Page Should Contain Checkbox</td><td>该关键字用来判断当前窗口页面是否包含了指定的 Checkbox 选项，接收[locator | message= | loglevel=INFO]三个参数，和 Page Should Contain Button 非常类似，locator 参数可以通过 id、name 来进行元素定位。示例：
<table>
<tr><td>Open Browser</td><td>https://passport.cnblogs.com/user/signin?ReturnUrl=https%3A%2F%2Fwww.cnblogs.com%2F</td><td>chrome</td></tr>
<tr><td>Page Should Contain Checkbox</td><td>id=remember_me</td><td>包含“下次自动登录”选项</td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Page Should Contain Element</td><td>该关键字用来判断当前窗口页面是否包含了指定的 Element 元素，接收[locator | message= | loglevel=INFO]三个参数，和上面的 Page Should Contain Checkbox 关键字非常类似，locator 参数可以通过 id、name 来进行元素定位。示例：
<table>
<tr><td>Page Should Contain Element</td><td>id= signin</td><td>包含了指定的元素</td></tr>
</table></td></tr>
<tr><td>Page Should Contain Image</td><td>该关键字用来判断当前窗口页面是否包含了指定的 Image 图片元素，接收[locator | message= | loglevel=INFO]三个参数，和上面的几个关键字类似，locator 参数可以通过 id、src、alt 来进行元素的定位。
【示例 1】locator 通过 src 来定位：
<table>
<tr><td>Open Browser</td><td>http://news.baidu.com/</td><td>chrome</td></tr>
<tr><td>Page Should Contain Image</td><td>https://box.bdimg.com/static/fisp_static/common/img/searchbox/logo_news_276_88_1f9876a.png</td><td>包含“指定的图片”</td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table>
【示例 2】locator 通过 alt 来定位：
<table>
<tr><td>Open Browser</td><td>http://news.baidu.com/</td><td>chrome</td></tr>
<tr><td>Page Should Contain Image</td><td>百度新闻</td><td>包含“指定的图片”</td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Page Should Contain Link</td><td>该关键字用来判断当前窗口页面是否包含了指定的 Link 元素，接收[locator | message= | loglevel=INFO]三个参数，和上面的几个关键字类似，locator 参数可以通过 id、name、href、link text 来进行元素的定位，这里以 href 的定位方式来作为示例：
<table>
<tr><td>Open Browser</td><td>http://www.baidu.com</td><td>chrome</td></tr>
<tr><td>Page Should Contain Link</td><td>http://news.baidu.com</td><td>包含“指定的 link”</td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Page Should Contain List</td><td>该关键字用来判断当前窗口页面是否包含了指定的 List，接收[locator | message= | loglevel=INFO]三个参数，locator 参数可以通过 id、name 来进行定位。示例：
<table>
<tr><td>Page Should Contain List</td><td>id=pf</td><td>包含正确</td></tr>
</table></td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Page Should Contain Radio Button</td><td>该关键字用来判断当前窗口页面是否包含了指定的被选中的按钮，接收[locator | message= | loglevel=INFO]三个参数，locator 参数可以通过 id、name、value 来进行定位。示例：<table><tr><td>Page Should Contain Radio Button</td><td>name=sw_f</td><td>包含正确</td></tr></table></td></tr>
<tr><td>Page Should Contain Textfield</td><td>该关键字用来判断当前窗口页面是否包含了指定的文本字段，接收[locator | message= | loglevel=INFO]三个参数，locator 参数可以通过 Id、name 来进行元素的定位。示例：<table><tr><td>Page Should Contain Textfield</td><td>name=syt</td><td></td></tr></table></td></tr>
<tr><td>Capture Page Screenshot</td><td>该关键字主要用来在自动化运行过程中对当前的屏幕进行截图，便于在自动化执行过程中遇到问题时通过截图的方式保存运行报错时的现场。该关键字接收[filename=None]一个参数，可以指定保存的文件的名称。示例：<table><tr><td>Capture Page Screenshot</td><td>Test.png</td></tr></table></td></tr>
<tr><td>Set Screenshot Directory</td><td>该关键字用来设置快照保存的目录，接收[path | persist=False]两个参数。可以和 Capture Page Screenshot 配合使用。示例：<table><tr><td>Set Screenshot Directory</td><td>d:\\</td></tr></table></td></tr>
<tr><td>Checkbox Should Be Selected</td><td>该关键字主要用来判断指定的 Checkbox 是否被选中，接收[locator]一个参数。locator 参数可以通过 id、name 来进行元素的定位。示例：<table><tr><td>Checkbox Should Be Selected</td><td>id=ppv1</td></tr></table></td></tr>
<tr><td>Clear Element Text</td><td>该关键字主要用来清空文本框中的内容，接收[locator]一个参数。示例：<table><tr><td>Clear Element Text</td><td>id=kw</td></tr></table></td></tr>
<tr><td>Click Image</td><td>该关键字用来模拟单击某一个图片，接收[locator]一个参数。locator 可以通过 id、src、alt 来进行定位。示例：<table><tr><td>Click Image</td><td>https://box.bdimg.com/static/fisp_static/common/img/searchbox/logo_news_276_88_1f9876a.png</td></tr></table></td></tr>
<tr><td>Close Window</td><td>该关键字用来关闭当前的窗口，不需要接收任何参数</td></tr>
<tr><td>Confirm Action</td><td>该关键字用来获取确认对话框中对应的确认信息，不需要接收任何参数，示例：<table><tr><td>Click Button</td><td>Delete</td></tr><tr><td>${confirmmessage}=</td><td>Confirm Action</td></tr><tr><td>log</td><td>${confirmmessage}</td></tr></table></td></tr>
<tr><td>Current Frame Contains</td><td>该关键用来判断当前的 Frame 中是否包含指定的文本内容，接收[text | loglevel=INFO]两个参数，示例：<table><tr><td>Current Frame Contains</td><td>指定的内容信息</td></tr></table></td></tr>
<tr><td>Double Click Element</td><td>该关键字用来模拟双击操作，接收[locator]一个关键字。locator 可以通过 id 或者 name 来进行定位。示例：<table><tr><td>Double Click Element</td><td>Id=iwe</td></tr></table></td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Drag And Drop</td><td>该关键字用来模拟拖拽操作，接收[source | target]两个参数，source 和 target 都是通过 locator 定位到具体元素。示例<table><tr><td>Drag And Drop</td><td>elementA</td><td>elementB</td></tr></table></td></tr>
<tr><td>Drag And Drop By Offset</td><td>该关键字同样是用来模拟拖曳操作，接收[source | xoffset | yoffset]三个参数，source 就是通过 locator 定位到的具体元素。该关键字的作用是将 source 元素拖曳到一个具有 x、y 坐标标志的位置。示例：<table><tr><td>Drag And Drop By Offset</td><td>element</td><td>100</td><td>-40</td></tr></table></td></tr>
<tr><td>Element Should Be Enabled</td><td>该关键字用来判断定位到的某个元素是否开启。该关键字接收[locator]一个参数，locator 可以通过 id 或者 name 来进行定位。示例：<table><tr><td>Element Should Be Enabled</td><td>id=pwa</td></tr></table></td></tr>
<tr><td>Element Should Be Visible</td><td>该关键字用来判断定位到的某个元素的属性是否是可见的。该关键字接收[locator | message=]两个参数，locator 可以通过 id 或者 name 来进行定位。示例：<table><tr><td>Element Should Be Visible</td><td>name=oot</td></tr></table></td></tr>
<tr><td>Element Should Contain</td><td>该关键字用来判断定位到的某个元素的内容中是否包含指定的内容。该关键字接收[locator | expected | message=]三个参数，locator 可以通过 id 或者 name 来进行定位。示例：<table><tr><td>Element Should Contain</td><td>id=pwa</td><td>包含的指定内容</td></tr></table></td></tr>
<tr><td>Element Text Should Be</td><td>该关键字用来判断定位到的某个元素中的文本内容是否和指定的期望的内容完全一致。该关键字接收[locator | expected | message=] 三个参数，locator 可以通过 id 或者 name 来进行定位，message 可以用来覆盖指定的提示信息。示例：<table><tr><td>Element Text Should Be</td><td>id=pwa</td><td>指定的内容</td></tr></table></td></tr>
<tr><td>Get Alert Message</td><td>该关键字用来获取一个 JavaScript alert 确认弹出框对应的提示信息。该关键字接收[dismiss=True]一个参数，如果不传入参数，默认 dismiss=True</td></tr>
<tr><td>Get Cookie Value</td><td>该关键字用来获取某个 Cookie 的值，接收[name]一个参数。这里的 name 参数指的是 Cookie 的名称。示例：<table><tr><td>Get Cookie Value</td><td>Book</td></tr></table></td></tr>
<tr><td>Input Password</td><td>该关键字和之前的 Input Text 关键字类似，不同的是由于密码具有私密性，因此在 Robot Framework 运行日志输出时不会直接输出密码到日志中。该关键字接收[locator | text]两个参数，locator 可以通过 id 或者 name 来进行元素定位，text 参数代表需要输入的密码。示例：<table><tr><td>Input Password</td><td>id=kw</td><td>123456</td></tr></table></td></tr>
<tr><td>Get Location</td><td>该关键字用于获取当前窗口页面访问的路径，不需要接收任何参数。示例：<table><tr><td>Open Browser</td><td>http://www.baidu.com</td><td>chrome</td></tr><tr><td>${Location}</td><td>Get Location</td><td></td></tr><tr><td>log</td><td>${Location}</td><td></td></tr><tr><td>Close Browser</td><td></td><td></td></tr></table></td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Get Matching Xpath Count</td><td>该关键字用来统计通过某个 xpath 匹配到具体元素的数量，在我们排除自动化错误时十分有效。通过一个 xpath 能匹配到多个元素时，经常会引起自动化报错，使用该关键字可以很快发现一个 xpath 是否会定位到多个元素。该关键字接收[xpath]一个参数。示例：
<table>
<tr><td>${count}</td><td>Get Matching Xpath Count</td><td>//*[@id="u1"]/a[1]</td></tr>
<tr><td>log</td><td>${count}</td><td></td></tr>
</table></td></tr>
<tr><td>Get Selected List Label</td><td>该关键字用来获取已经选择的下拉列表的 Label 标签值，接收[locator]一个参数。locator 可以通过 id 或者 name 来进行元素的定位。示例：
<table>
<tr><td>Open Browser</td><td>http://tieba.baidu.com/f/search/adv?red_tag=u3387165643</td><td>chrome</td></tr>
<tr><td>${label}</td><td>Get Selected List Label</td><td>name=sm</td></tr>
<tr><td>log</td><td>${label}</td><td></td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table>
运行结果如下：
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0024
20180728 23:47:47.576 : INFO : Opening browser 'chrome' to base url 'http://tieba.baidu.com/f/search/adv?red_tag=u3387165643'
20180728 23:47:55.213 : INFO : get_attribute multiple
20180728 23:47:55.290 : INFO : ${label} = 按时间倒序
20180728 23:47:55.292 : INFO : 按时间倒序
Ending test: RobotFrameworkTest1.TestSuite6.TestCase0024</td></tr>
<tr><td>Get Selected List Labels</td><td>该关键字和 Get Selected List Label 类似，不同的是该关键用来获取已经选择的多个列表的标签值，适合于多选的情况。该关键字接收[locator]一个参数，locator 可以通过 id 或者 name 来进行元素的定位。示例：
<table>
<tr><td>${labels}</td><td>Get Selected List Labels</td><td>ame=sm</td></tr>
<tr><td>log</td><td>${labels}</td><td></td></tr>
</table></td></tr>
<tr><td>Get Selected List Values</td><td>该关键字和 Get Selected List Value 关键字类似，不同的是 Get Selected List Values 适合获取多选下拉列表框中的 Value 值。该关键字接收[locator]一个参数，locator 可以通过 id 或者 name 等来进行元素定位。示例：
<table>
<tr><td>Open Browser</td><td>http://tieba.baidu.com/f/search/adv?red_tag=u3387165643</td><td>chrome</td></tr>
<tr><td>@{Value}</td><td>Get Selected List Values</td><td>name=sm</td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table>
运行结果如下：
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0026
20180729 00:07:12.416 : INFO : Opening browser 'chrome' to base url 'http://tieba.baidu.com/f/search/adv?red_tag=u3387165643'
20180729 00:07:21.235 : INFO : get_attribute multiple
20180729 00:07:21.312 : INFO : get_attribute value
20180729 00:07:21.320 : INFO : @{Value} = [1]
Ending test: RobotFrameworkTest1.TestSuite6.TestCase0026</td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Get Selenium Speed</td><td>该关键字用来获取 Selenium 运行的速度，在做自动化框架运行性能监控以及性能调优分析时非常有用，不需要接收任何参数。示例：
<table>
<tr><td>${speed}</td><td>Get Selenium Speed</td></tr>
<tr><td>log</td><td>${speed}</td></tr>
</table></td></tr>
<tr><td>Get Selenium Timeout</td><td>该关键字用来获取 Selenium 运行出现超时的超时时间，在做自动化框架运行性能监控以及性能调优分析时非常有用，不需要接收任何参数。示例：
<table>
<tr><td>${time}</td><td>Get Selenium Timeout</td></tr>
<tr><td>log</td><td>${time}</td></tr>
</table></td></tr>
<tr><td>Get Window Position</td><td>该关键字用来获取当前页面窗口的坐标位置，先返回横坐标，再返回纵坐标，不需要接收任何参数。示例：
<table>
<tr><td>Open Browser</td><td>http://tieba.baidu.com/f/search/adv?red_tag=u3387165643</td><td>chrome</td></tr>
<tr><td>${x}</td><td>${y}=</td><td>Get Window Position</td></tr>
<tr><td>log</td><td>${x}</td><td></td></tr>
<tr><td>log</td><td>${y}</td><td></td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Get Window Size</td><td>该关键字用来获取当前页面窗口的大小，会返回窗口的宽和高两个值，先返回宽度，再返回高度，不需要接收任何参数。示例：
<table>
<tr><td>Open Browser</td><td>http://tieba.baidu.com/f/search/adv?red_tag=u3387165643</td><td>chrome</td></tr>
<tr><td>${width}</td><td>${height}</td><td>Get Window Size</td></tr>
<tr><td>log</td><td>${width}</td><td></td></tr>
<tr><td>log</td><td>${height}</td><td></td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table>
运行结果如下：
<pre>
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0028
20180729 00:25:13.981 : INFO : Opening browser 'chrome' to base url
'http://tieba.baidu.com/f/search/adv?red_tag=u3387165643'
20180729 00:25:23.873 : INFO : ${width} = 1050
20180729 00:25:23.873 : INFO : ${height} = 840
20180729 00:25:23.875 : INFO : 1050
20180729 00:25:23.876 : INFO : 840
Ending test: RobotFrameworkTest1.TestSuite6.TestCase0028
</pre></td></tr>
<tr><td>List Windows</td><td>该关键字用来获取当前打开的所有页面窗口，不需要接收任何参数，返回的结果是一个列表的形式。示例：
<table>
<tr><td>Open Browser</td><td>http://tieba.baidu.com/f/search/adv?red_tag=u3387165643</td><td>chrome</td></tr>
<tr><td>@{result}</td><td>List Windows</td><td></td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table></td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Mouse Down</td><td>该关键字用来模拟按下鼠标的左键操作，接收[locator]一个参数，locator 可以通过 id 和 name 来进行元素的定位。示例：
<table><tr><td>Mouse Down</td><td>id=kw</td></tr></table></td></tr>
<tr><td>Mouse Up</td><td>该关键字和 Mouse Down 刚好相反，用来释放按下的鼠标左键。该关键字接收[locator]一个参数，locator 可以通过 id 和 name 来进行元素的定位。示例：
<table><tr><td>Mouse Up</td><td>id=kw</td></tr></table></td></tr>
<tr><td>Mouse Down On Image</td><td>该关键字用来模拟在页面上的一张图片上按下鼠标的左键操作，该关键字接收[locator]一个参数，locator 可以通过 id、src、alt 来进行元素的定位。示例：
<table><tr><td>Mouse Down On Image</td><td>https://box.bdimg.com/static/fisp_static/common/img/searchbox/logo_news_276_88_1f9876a.png</td></tr></table></td></tr>
<tr><td>Mouse Down On Link</td><td>该关键字用来模拟在页面上的一个链接上按下鼠标左键的操作。该关键字接收[locator]一个参数，locator 可以通过 id、href、link text 来进行元素的定位。示例：
<table><tr><td>Mouse Down On Link</td><td>http://map.baidu.com</td></tr></table></td></tr>
<tr><td>Mouse Out</td><td>该关键字用来模拟鼠标离开页面上的一个元素的操作。该关键字接收[locator]一个参数，locator 可以通过 id、name 来进行元素的定位。示例：
<table>
<tr><td>Open Browser</td><td>https://www.baidu.com/</td><td>chrome</td></tr>
<tr><td>Mouse down</td><td>id=kw</td><td></td></tr>
<tr><td>Mouse Out</td><td>id=kw</td><td></td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>Reload Page</td><td>该关键字用来模拟重新加载当前窗口的页面，不需要接收任何参数</td></tr>
<tr><td>Register Keyword To Run On Failure</td><td>该关键字用来模拟自动化案例执行失败时需要执行的操作。比如某个案例执行失败后浏览器并没有关闭，此时需要在失败时将浏览器关闭，就可以用这个关键字来操作。该关键字接收[keyword]一个参数，keyword 表示执行失败时需要执行的关键字操作，就是我们常说的容错操作。示例：
<table>
<tr><td>Open Browser</td><td>https://www.baidu.com/</td><td>chrome</td></tr>
<tr><td>Register Keyword To Run On Failure</td><td>Close Browser</td><td></td></tr>
<tr><td>Mouse down</td><td>id=kw111</td><td></td></tr>
<tr><td>Close Browser</td><td></td><td></td></tr>
</table>
运行结果如下：
<pre>
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0030
20180729 09:31:02.645 : INFO : Opening browser 'chrome' to base url 'https://www.baidu.com/'
20180729 09:31:10.295 : INFO : Close Browser will be run on failure.
20180729 09:31:10.296 : INFO : Simulating Mouse Down on element 'id=kw111'
20180729 09:31:11.472 : FAIL : ERROR: Element id=kw111 not found.
Ending test: RobotFrameworkTest1.TestSuite6.TestCase0030
</pre>
从运行结果看，当通过 id=kw111 定位不到对应的元素时执行就失败了，但是由于我们在第二步中做了 Register Keyword To Run On Failure 容错操作，所以就算执行失败了也可以将浏览器正常关闭掉</td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Select All From List</td><td>该关键字用来模拟选择表单中的所有列表。该关键字接收[locator]一个参数，locator 可以通过 id、name 来进行元素的定位。示例：<table><tr><td>Select All From List</td><td>id=aac</td></tr></table></td></tr>
<tr><td>Select Checkbox</td><td>该关键字用来模拟选择一个 Checkbox 的操作。该关键字接收[locator]一个参数，locator 可以通过 id、name 来进行元素的定位。示例：<table><tr><td>Select Checkbox</td><td>id=box</td></tr></table></td></tr>
<tr><td>Select From List By Index</td><td>该关键字用来模拟通过下拉列表的 Index 来选中指定的下拉列表的选项，该关键字接收[locator | *indexes]多个参数，locator 可以通过 id、name 来进行元素的定位，indexes 可以允许传入多个。示例：<table><tr><td>Select From List By Index</td><td>name=sm</td><td>0</td></tr></table></td></tr>
<tr><td>Select From List By Label</td><td>该关键字用来模拟通过下拉列表的 Label 来选中指定的下拉列表的选项。该关键字接收[locator | *labels]多个参数，locator 可以通过 id、name 来进行元素的定位，labels 可以允许传入多个。示例：<table><tr><td>Select From List By Label</td><td>name=sm</td><td>按时间顺序</td></tr></table></td></tr>
<tr><td>Select From List By Value</td><td>该关键字用来模拟通过下拉列表的 Value 值来选中指定的下拉列表的选项，Value 值可以通过浏览器的开发者工具操作来获得。该关键字接收[locator | *values]多个参数，locator 可以通过 id、name 来进行元素的定位，values 可以允许传入多个。示例：<table><tr><td>Select From List By Value</td><td>name=sm</td><td>1</td></tr></table></td></tr>
<tr><td>Select Window</td><td>Select Window 关键字用来模拟打开了多个页面窗口时在不同的窗口之间进行窗口切换的操作。该关键字接收[locator=None]一个参数，locator 可以是 name、窗口 title、url、window handle 等。示例：<table><tr><td>Open Browser</td><td>http://www.baidu.com</td><td>chrome</td></tr><tr><td>Click Link</td><td>name=tj_trmap</td><td></td></tr><tr><td>Select Window</td><td>百度地图</td><td></td></tr><tr><td>Close Browser</td><td></td><td></td></tr></table></td></tr>
<tr><td>Set Selenium Timeout</td><td>该关键字用来设置一个 Selenium 操作时的超时时间，避免一直循环等待，导致其他的测试用例无法继续执行，接收[seconds]一个参数。示例：<table><tr><td>Set Selenium Timeout</td><td>5</td></tr></table></td></tr>
<tr><td>Submit Form</td><td>该关键字用来模拟表单的提交操作，接收[locator=None]一个参数。示例：<table><tr><td>Submit Form</td><td>id=form1</td></tr></table></td></tr>
<tr><td>Switch Browser</td><td>该关键字用来模拟打开了多个浏览器后在多个浏览器中的切换操作，接收[index_or_alias]一个参数。示例：<table><tr><td>Open Browser</td><td>http://www.baidu.com</td><td>chrome</td></tr><tr><td>Location Should Be</td><td>https://www.baidu.com</td><td></td></tr><tr><td>Open Browser</td><td>https://www.cnblogs.com/</td><td>ie</td></tr><tr><td>Switch Browser</td><td>1</td><td></td></tr><tr><td>Close All Browsers</td><td></td><td></td></tr></table></td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Table Cell Should Contain</td><td>该关键字用来判断表格中指定的字段中是否包含期望的内容。该关键字接收[table_locator | row | column | expected | loglevel=INFO]五个参数，其中 row 和 column 参数都是从 1 开始的。示例：<table><tr><td>Table Cell Should Contain</td><td>tableelement</td><td>1</td><td>1</td><td>robot</td></tr></table></td></tr>
<tr><td>Table Header Should Contain</td><td>该关键字用来判断表格中的表头是否包含指定的内容，接收[table_locator | expected | loglevel=INFO]三个参数。示例：<table><tr><td>Table Header Should Contain</td><td>tableelement</td><td>robot</td></tr></table></td></tr>
<tr><td>Wait Until Element Contains</td><td>该关键字用来模拟等待页面中加载到指定的内容时才做后面的操作，防止页面窗口没有完全加载完成就执行接下来的关键字操作，从而导致操作失败。该关键字接收[locator | text | timeout=None | error=None]四个参数。示例：<table><tr><td>Wait Until Element Contains</td><td>id=kw</td><td>robot</td><td>5</td></tr></table></td></tr>
<tr><td>Xpath Should Match X Times</td><td>该关键字用来判断某个 xpath 路径可以匹配到的次数，是一个断言关键字，接收[xpath | expected_xpath_count | message= | loglevel=INFO]四个参数，示例：<table><tr><td>Xpath Should Match X Times</td><td>//*[@id="form"]/span[1]/span</td><td>1</td></tr></table></td></tr>
</table>

5.2 SikuliLibrary 库的使用

5.2.1 Sikuli 简介

Sikuli 是一种图形化编程技术，或者也可以说是一种图形化的自动化测试工具，平时在屏幕上看到的任何画面，Sikuli 都可以使用图像识别的方式来进行操作。Sikuli 不需要像 WebDriver 那样通过查找元素的方式去对页面进行定位，而是使用屏幕截图的方式来定位页面的按钮等，Sikuli 用于自动化测试的优点如下：

- 可以测试不易识别或无法定位的对象，比如地图 、Flash 和图表等。
- 可以验证和识别图片。
- 直接对图片进行操作，更加通俗易懂，容易维护。
- 适用于 Window/Linux/Mac OS X 桌面应用，甚至是 iPhone 和 Android 模拟器的自动化测试。

Sikuli 同样也有很大的缺点：

- 对于相似的图片或者按钮容易识别错误。
- 由于只能对图像进行操作，因此不够灵活，一般只能用于辅助测试，比较适合于辅助 WebDriver 进行 Web 自动化测试。

通过访问 https://github.com/sikuli/sikuli 网址可以进入 sikuli 的 GitHub 中获取最新版本的 sikuli，如图 5-2-1 所示。

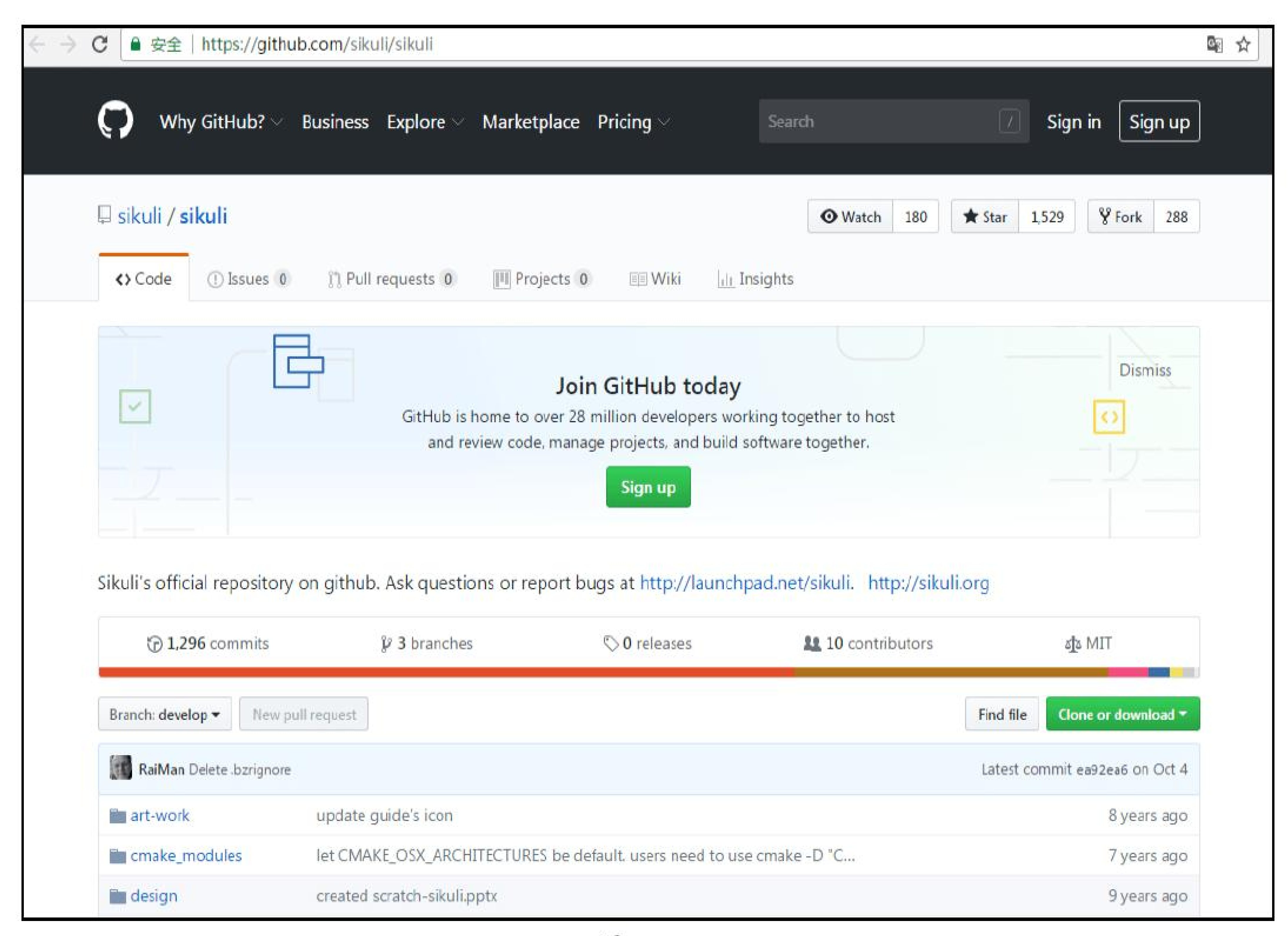


图 5-2-1

5.2.2　SikuliLibrary 的使用

通过访问 GitHub 地址 https://github.com/rainmanwy/robotframework-SikuliLibrary 可以下载和安装 robotframework-SikuliLibrary 库，如图 5-2-2 所示。

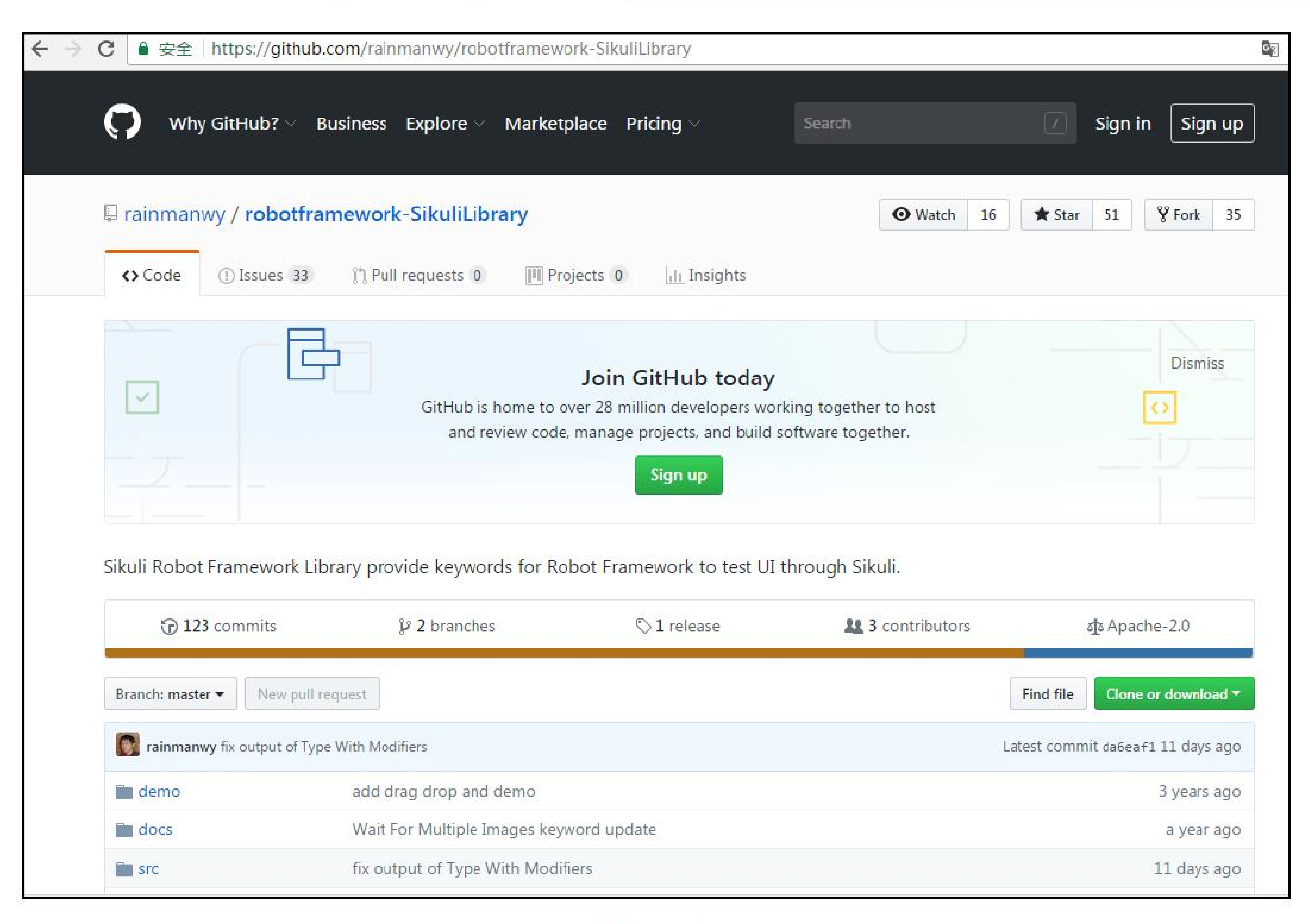


图 5-2-2

也可以通过 pip install robotframework-SikuliLibrary 来进行在线安装。安装完成后，在使用时需要在 RIDE 中导入 SikuliLibrary 库，如图 5-2-3 所示。

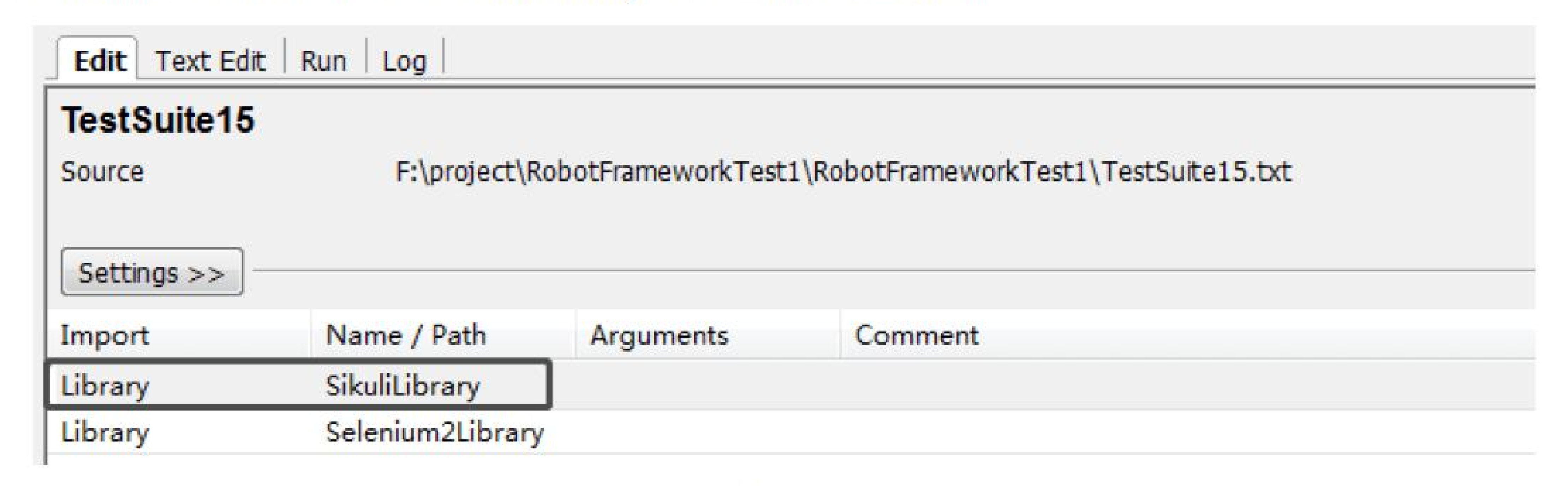


图 5-2-3

【示例 1】使用 WebDriver 和 sikuli 结合的方式来模拟单击百度首页进行搜索。

截取百度首页的搜索图标的图片，并且放入指定的磁盘位置，比如放入到本地电脑磁盘 D 盘的根目录下，如图 5-2-4 所示，然后在使用 Click 关键字时将传入 D 盘根目录下的 a.png 的路径作为该关键字的参数。



图 5-2-4

使用 SikuliLibrary 中的 Click 关键字来模拟单击百度首页的 Bai百度 图标，然后看一下单击后的效果，如图 5-2-5 所示

Open Browser	http://www.baidu.com	chrome
Click	d:\\a.png	
Close Browser		

图 5-2-5

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite15.TestCase0001
20181202 17:59:37.594 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20181202 17:59:52.316 :  INFO : Params: [d:\a.png]
20181202 17:59:52.316 :  INFO :
<img src='sikuli_captured/sikuliximage-1543744791279.png'/>
[log] CLICK on L(534,294)@S(0)[0,0 1600x900] (524 msec)
Ending test:  RobotFrameworkTest1.TestSuite15.TestCase0001
```

从运行结果可以看到，通过 Click　d:\\a.png 的方式执行的效果和通过 WebDriver 中的 xpath= //*[@id="lg"]/map/area 执行的效果完全一样，如图 5-2-6 和图 5-2-7 所示。



图 5-2-6



图 5-2-7

【示例 2】使用 SikuliLibrary 中的 Get Match Score 关键字来从屏幕上获取指定目标图像的匹配度，这里依旧采用【示例 1】中的图像作为待匹配的目标图像，如图 5-2-8 所示。

Open Browser	http://www.baidu.com	chrome
${score}	Get Match Score	d:\\a.png
log	${score}	

图 5-2-8

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite15.TestCase0002
20181202 20:39:56.115 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20181202 20:40:09.648 :  INFO : Params: [d:\a.png]
20181202 20:40:09.648 :  INFO : <img
src='sikuli_captured/sikuliximage-1543754409621.png'/>
20181202 20:40:09.648 :  INFO : ${score} = 1.0
20181202 20:40:09.649 :  INFO : 1.0
Ending test:   RobotFrameworkTest1.TestSuite15.TestCase0002
```

从运行结果可以看到，由于屏幕上的百度首页中存在和目标图片一致的图像，所以最终获取到的匹配度为 1.0，说明为完全匹配，如图 5-2-9 所示。



图 5-2-9

5.2.3　SikuliLibrary 的工作原理

如图 5-2-10 所示，Robot Framework 的 SikuliLibrary 库和 Sikuli 之间的通信关系如下，借助 Robot Framework 提供的 XML-RPC 协议的 Remote 服务进行通信，通过 Remote 调用方式来连接用 Java 语言实现的 Sikuli API 操作，正是由于有了 XML-RPC 协议的远程调用，使得不管是 Python 语言、Java 语言或者其他语言都可以调用 Sikuli 的 Java API。关于 Remote 的调用方式，在后面还会继续详细讲解。

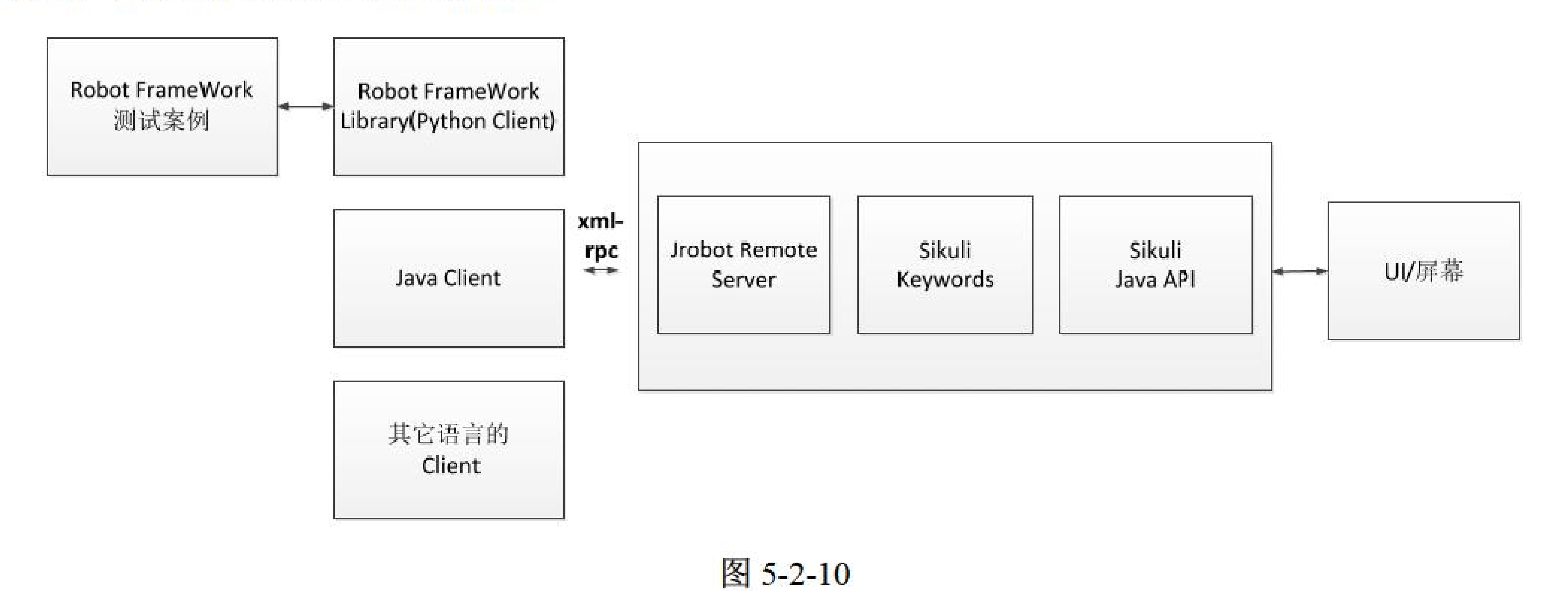


图 5-2-10

另外，从 SikuliLibrary 的源码中我们也可以分析出其调用的方式，在 SikuliLibrary 的 Python

源码 sikuli.py 中有如下几个重要的 Python 函数， __init__函数中定义了初始化操作，并且指定了默认 mode 为 OLD 模式，OLD 模式下 Sikuli 对应的 Java API 进程会在 init 初始化的时候跟随一起启动，start_sikuli_process 函数中定义 Sikuli Java API 进程的启动过程，从源码中可以看到，启动时是通过执行 java –jar SikuliLibrary.jar 命令的方式来启动 SikuliLibrary 的 Java API 进程，通过 connect_remote_library 函数来连接启动好的 Java Remote 服务，然后通过 run_keyword 函数来执行 Ride 上传入的关键字操作。

sikuli.py 的部分源码如下：

```
......
   def __init__(self, port=0, timeout=3.0, mode='OLD'):
       """
       @port: sikuli java process socket port
       @timeout: Timeout of waiting java process started
       @mode: if set as 'DOC',  will stop java process automatically,
             if set as 'PYTHON', means library is running out of robot environment
             if set as 'CREATE', it is only for mvn package usage, will create
keywords.py file
             if set as 'OLD'(default), sikuli java process will be started when
library is inited
             if set as 'NEW', user should use 'start_sikuli_process' to start java
process
       """
       self.logger = self._init_logger()
       self.timeout = float(timeout)
       self.port = None
       self.remote = None
       self.mode = mode.upper().strip()
       if mode == 'OLD':
          self.start_sikuli_process(port)
       if mode.upper().strip() == 'DOC':
          self.start_sikuli_process()
          self._stop_thread(4)
       elif mode.upper().strip() == 'PYTHON':
          self.connect_sikuli_process(port)
       elif mode.upper().strip() == 'CREATE':
          self._create_keywords_file()
       elif mode.upper().strip() != 'NEW':
          self._check_robot_running()

   def start_sikuli_process(self, port=None):
       """
       This keyword is used to start sikuli java process.
       If library is inited with mode "OLD", sikuli java process is started
automatically.
       If library is inited with mode "NEW", this keyword should be used.

       :param port: port of sikuli java process, if value is None or 0, a random
free port will be used
       :return: None
       """
       if port is None or int(port) == 0:
          port = self._get_free_tcp_port()
       self.port = port
       start_retries = 0
       started = False
       while start_retries < 5:
```

```
            try:
                self._start_sikuli_java_process()
            except RuntimeError as err:
                print('error........%s' % err)
                if self.process:
                    self.process.terminate_process()
                self.port = self._get_free_tcp_port()
                start_retries += 1
                continue
            started = True
            break
        if not started:
            raise RuntimeError('Start sikuli java process failed!')
        self.remote = self._connect_remote_library()
......
    def _connect_remote_library(self):
        remoteUrl = 'http://127.0.0.1:%s/' % str(self.port)
        remote = Remote(remoteUrl)
        self._test_get_keyword_names(remote)
        return remote
......
    def _start_sikuli_java_process(self):
        libFolder = os.path.join(os.path.abspath(os.path.dirname(__file__)),
'lib')
        jarList = glob.glob(libFolder + os.sep + '*.jar')
        if len(jarList) != 1:
            raise Exception('Sikuli jar package should be exist in lib folder')
        sikuliJar = jarList[0]
        java = 'java'
        arguments = ['-jar', sikuliJar, str(self.port),
self._get_output_folder()]
        self.process = Process()
        if os.getenv("DISABLE_SIKULI_LOG"):
            self.process.start_process(java, *arguments, shell=True)
        else:
            self.process.start_process(java, *arguments, shell=True,
stdout=self._output_file(),
                                       stderr=self._err_file())
        self.logger.info('Start sikuli java process on port %s' % str(self.port))
        self._wait_process_started()
        self.logger.info('Sikuli java process is started')
......

    def run_keyword(self, name, arguments=[], kwargs={}):
        if name == 'start_sikuli_process':
            return self.start_sikuli_process(*arguments)
        return self.remote.run_keyword(name, arguments, kwargs)
......
```

5.2.4　SikuliLibrary 常用关键字介绍

表 5-2-1 介绍了 SikuliLibrary 库中常用关键字的用法。

表 5-2-1　SikuliLibrary 库中常用关键字的用法

关键字	使用描述
Click	该关键字用于模拟单击屏幕中指定的图像，接收[image | xOffset=0 | yOffset=0]三个参数，image 参数指的是图片的实际位置，xOffset 和 yOffset 为指定图片的坐标位置，如果不传入的话，默认为(0,0)
Click In	该关键字用于模拟在指定的目标图像上单击指定区域的图像，接收[areaImage | targetImage]两个参数，targetImage 参数指的是目标图片，areaImage 参数指的是目标图像上指定的图像区域，这两个参数都是传入图片的实际路径
Double Click	该关键字和 Click 关键字类似，不同的是，这个关键字模拟的是对屏幕上指定的图像做双击操作
Double Click In	该关键字和 Click In 关键字类似，不同的是，该关键字模拟的是对目标图像上指定图像区域做双击操作
Drag And Drop	该关键字模拟的是拖动操作，接收[srcImage | targetImage]两个参数，模拟从源头像(srcImage)拖动到目标图像(targetImage)，参数 srcImage 和 targetImage 都是传入图像的实际路径
Drag And Drop By Offset	该关键字和 Drag And Drop 关键字很类似，不同的是，Drag And Drop By Offset 是通过 offset.的方式将源头像拖动到指定的目标区域，该关键字接收[srcImage|xOffset|yOffset]三个参数，参数 srcImage 指的是源头像的实际路径，参数 xOffset 和 yOffset 指的是拖动到的目标区域的坐标位置
Exists	该关键字用于断言在屏幕上是否存在指定的图像，该关键字接收[image | timeout=]两个参数，image 指的是图像的实际路径，timeout 是超时时间，超过指定超时时间后，会返回失败
Get Current Screen Id	该关键字用于获取当前屏幕的 id，SikuliLibrary 中对屏幕进行编号
Capture Screen	该关键字用于抓屏
Get Match Score	该关键字用于从屏幕上获取指定图像的匹配度，该关键字接收[image]一个参数，image 指的是图片的实际路径
Right Click	该关键字用于模拟鼠标右击操作，接收[image]一个参数，image 指的是图片的实际路径
Right Click In	该关键字用户模拟在指定目标图像区域的指定图像区域上做鼠标右击操作，接收[areaImage | targetImage]两个参数，targetImage 参数指的目标图片，areaImage 参数指的是目标图像上指定的图像区域，这两个参数都是传入图片的实际路径
Screen Should Contain	该关键字用于判断屏幕上是否存在指定的图像，接收[image]一个参数，image 参数需要传入图片的实际路径
Set Capture Folder	该关键字用于设置获取到的图片的路径，接收[path]一个参数

第 6 章 编写自定义的 Robot Framework Lib

Robot Framework 官网提供了很多 Lib。除了官网中已经有的 Lib 外，我们还可以自己来编写 Lib，既可以使用 Python 语言来编写，也可以使用 Java 语言或者其他的编程语言来编写自定义的 Robot Framework Lib。

6.1 使用 Python 编写自定义的 Robot Framework Lib

6.1.1 使用 Python 构建 Lib 工程

可以用来开发 Python Lib 的 IDE 工具很多，常见的有 PyCharm、Eclipse with PyDev 插件等。在 Robot Framework 官网中已经提供了 RobotFramework-EclipseIDE 插件，访问地址为 https://github.com/NitorCreations/RobotFramework-EclipseIDE，可以直接下载，如图 6-1-1 所示，它用来支持 Eclipse。

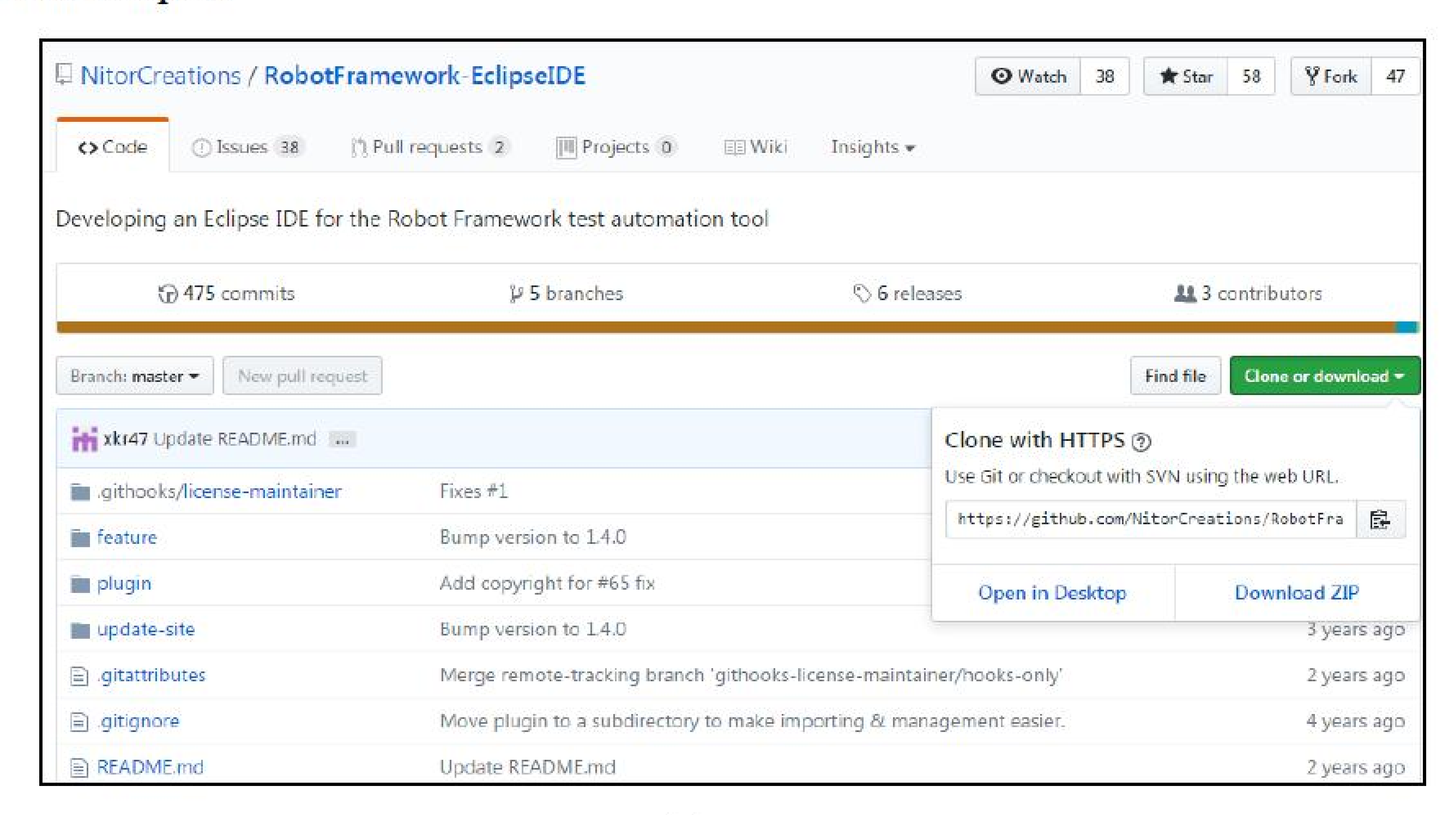


图 6-1-1

在这里我们以 Eclipse with PyDev 插件的形式来构建一个 Lib，既可以从 http://www.pydev.org/下载对应的插件，也可以通过 Eclipse 在线安装的方式进行安装。在线安装的地址为 http://www.pydev.org/updates，如图 6-1-2 所示。

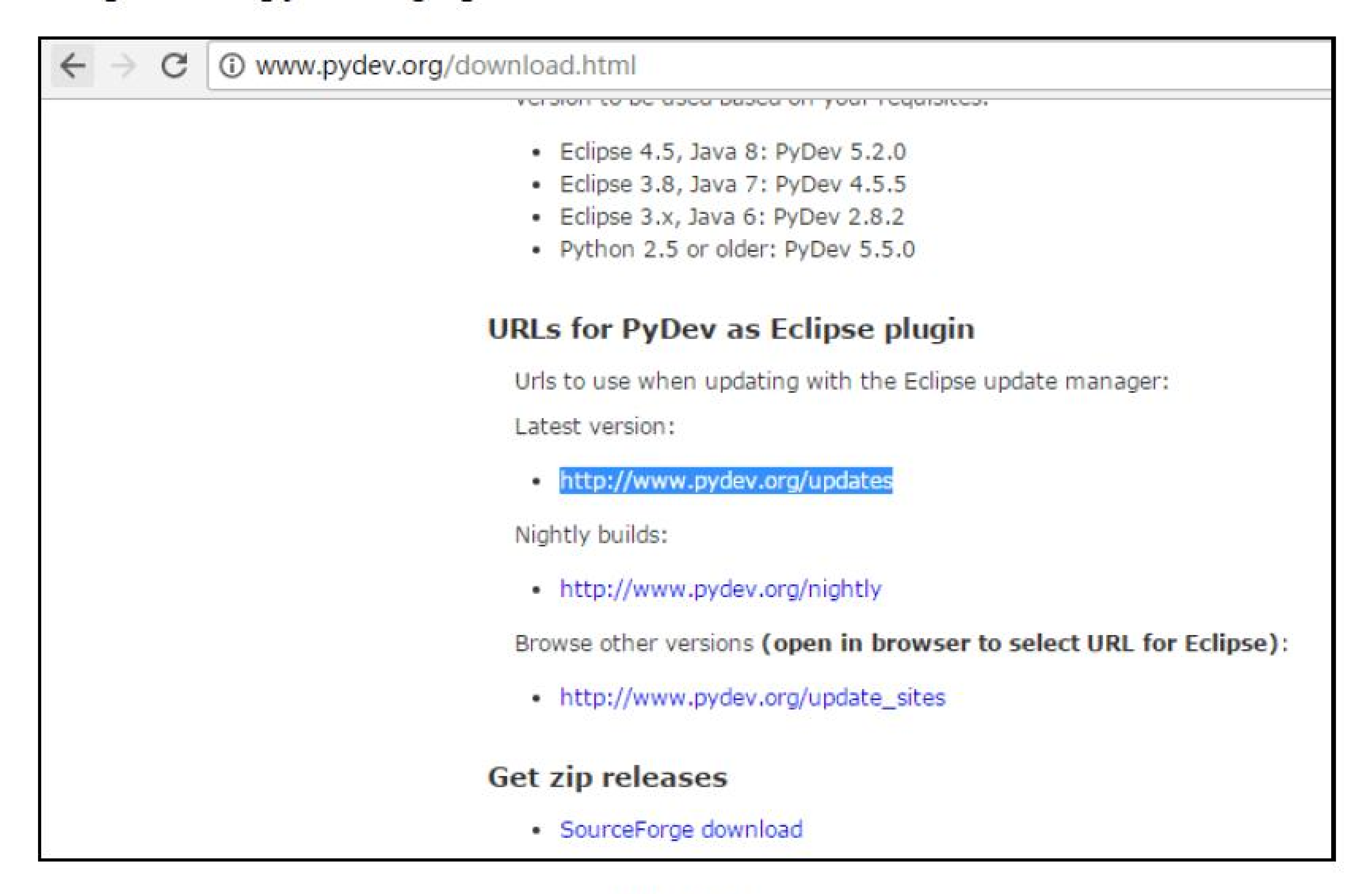


图 6-1-2

启动 Eclipse 后，单击 Eclipse 菜单 Help→Install New Software...，在弹出的对话框中单击 Add 按钮，接着在 Name 文本框中输入“Pydev”，在 Location 文本框中输入“http://pydev.org/updates”，如图 6-1-3 和图 6-1-4 所示。

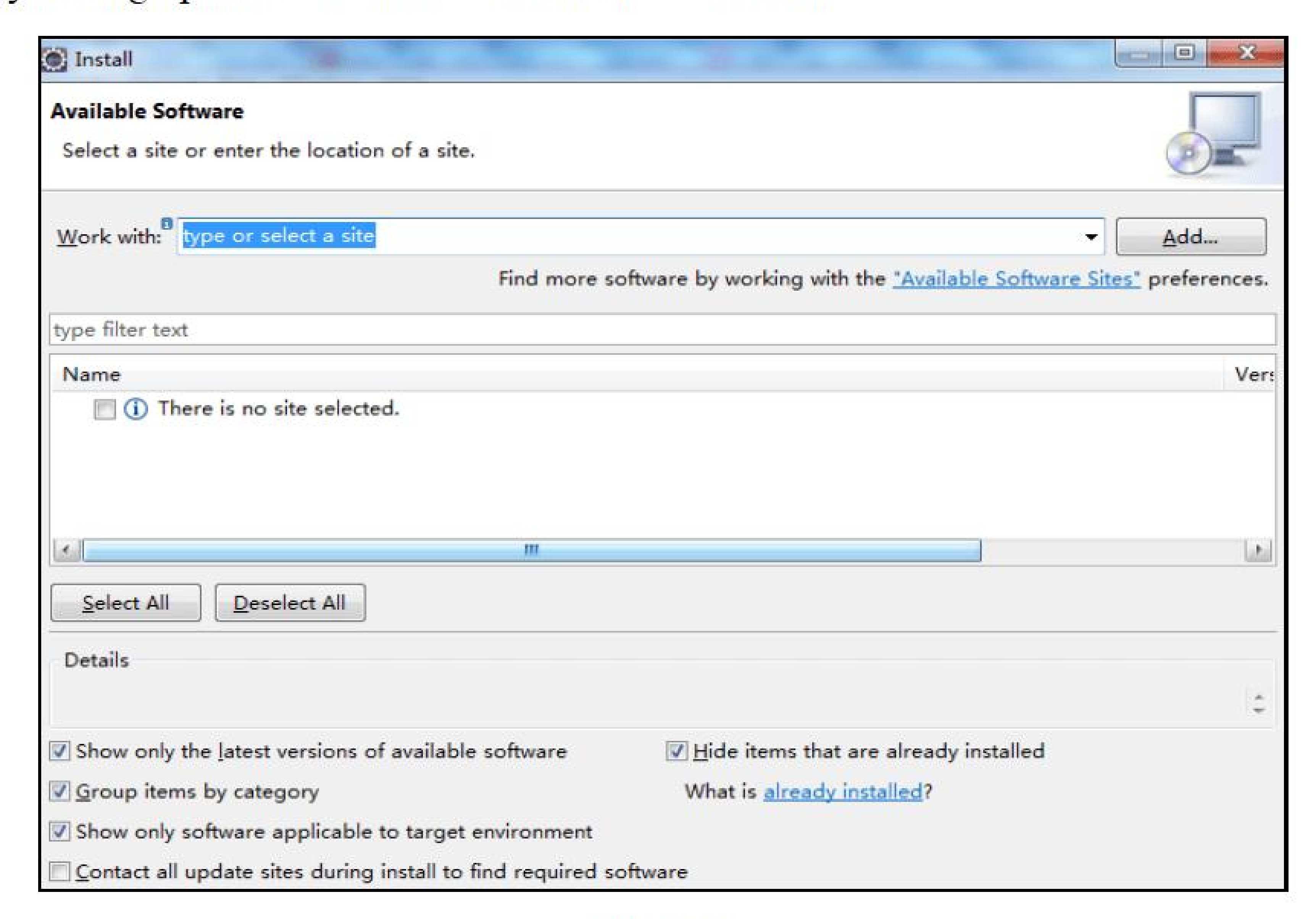


图 6-1-3

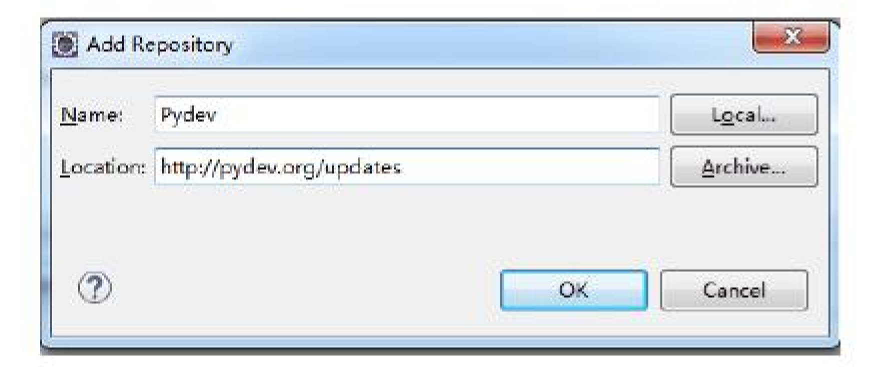


图 6-1-4

单击 OK 按钮后，可以看到供安装的插件选项，这里我们选择全部安装，如图 6-1-5 所示。

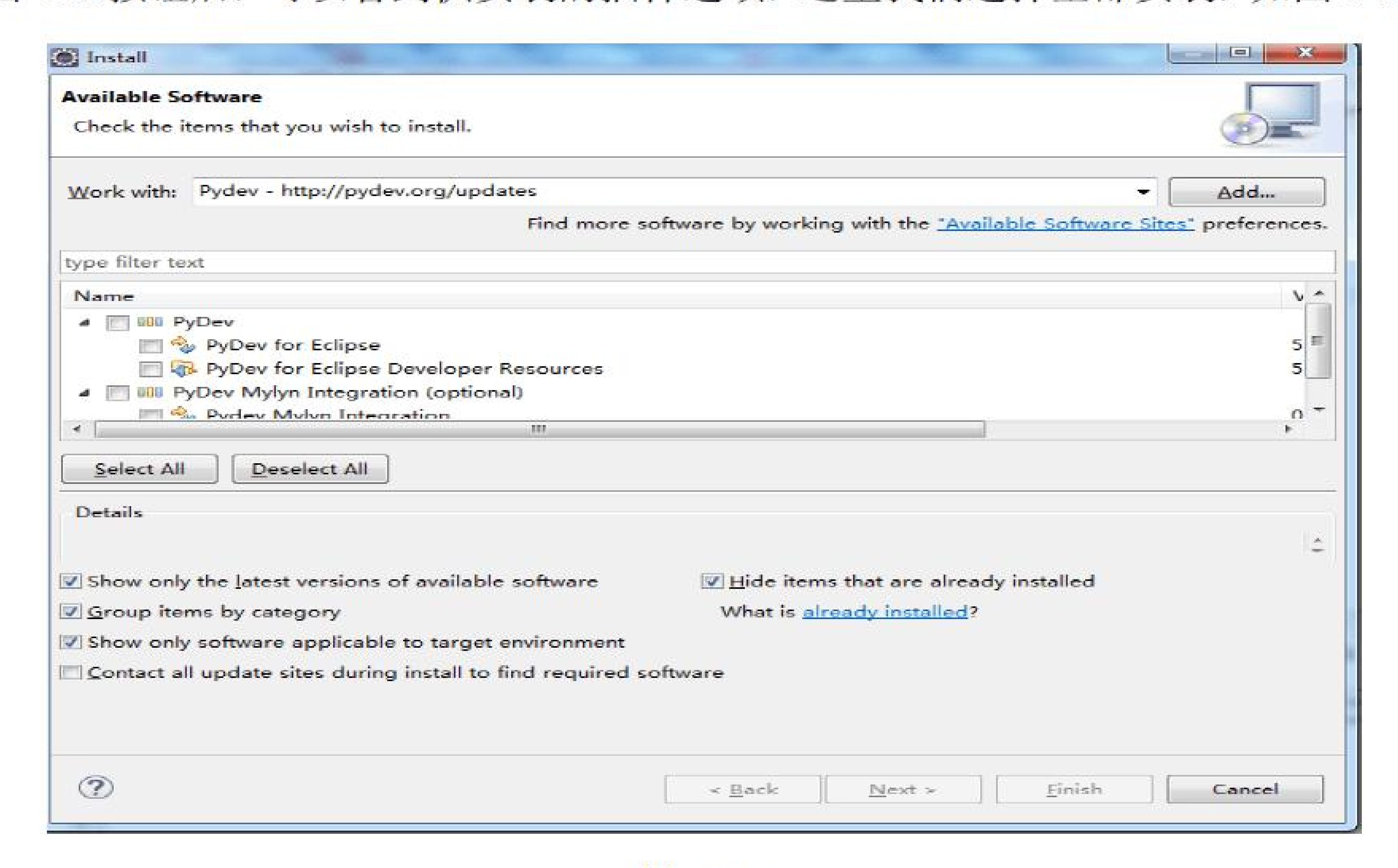


图 6-1-5

然后单击 Next 按钮，等待安装完成即可，如图 6-1-6 和图 6-1-7 所示。

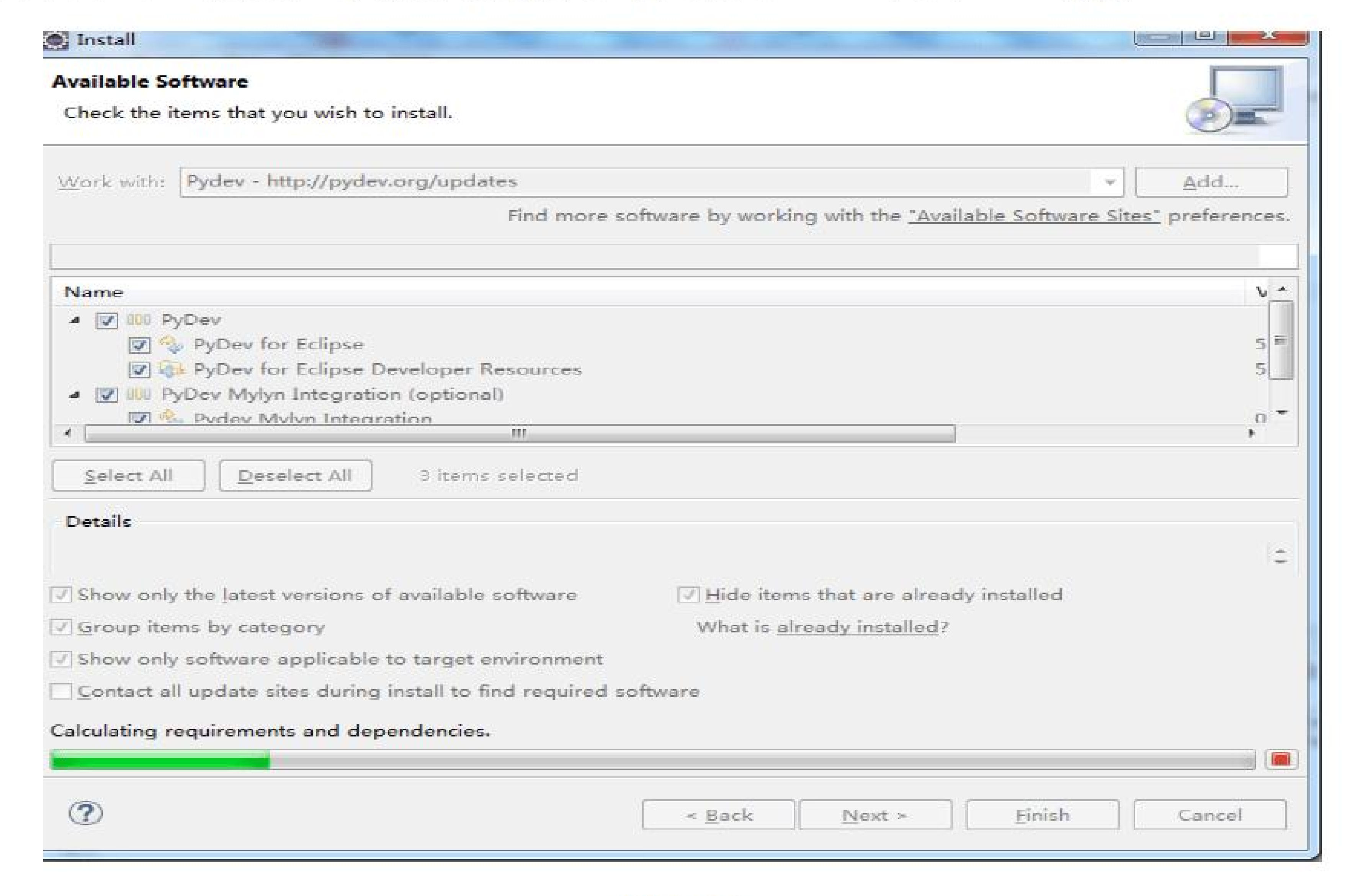


图 6-1-6

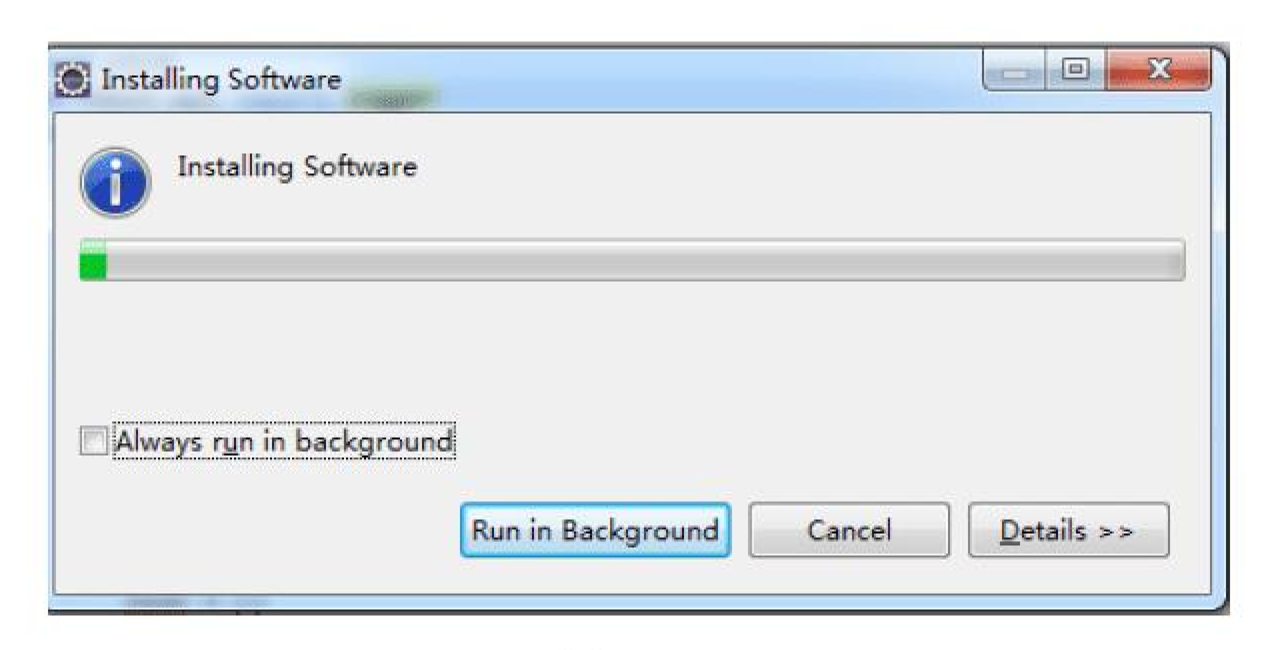


图 6-1-7

安装完成后，需要在 Eclipse 中配置 Python 解释器。在 Eclipse 菜单栏中，单击 Windows→Preferences，在对话框中选择 pyDev→Interpreter→Python Interpreter，单击 New 按钮。选择 python.exe 的路径，打开后显示出一个包含很多复选框的窗口，单击 OK 按钮，如图 6-1-8 所示。

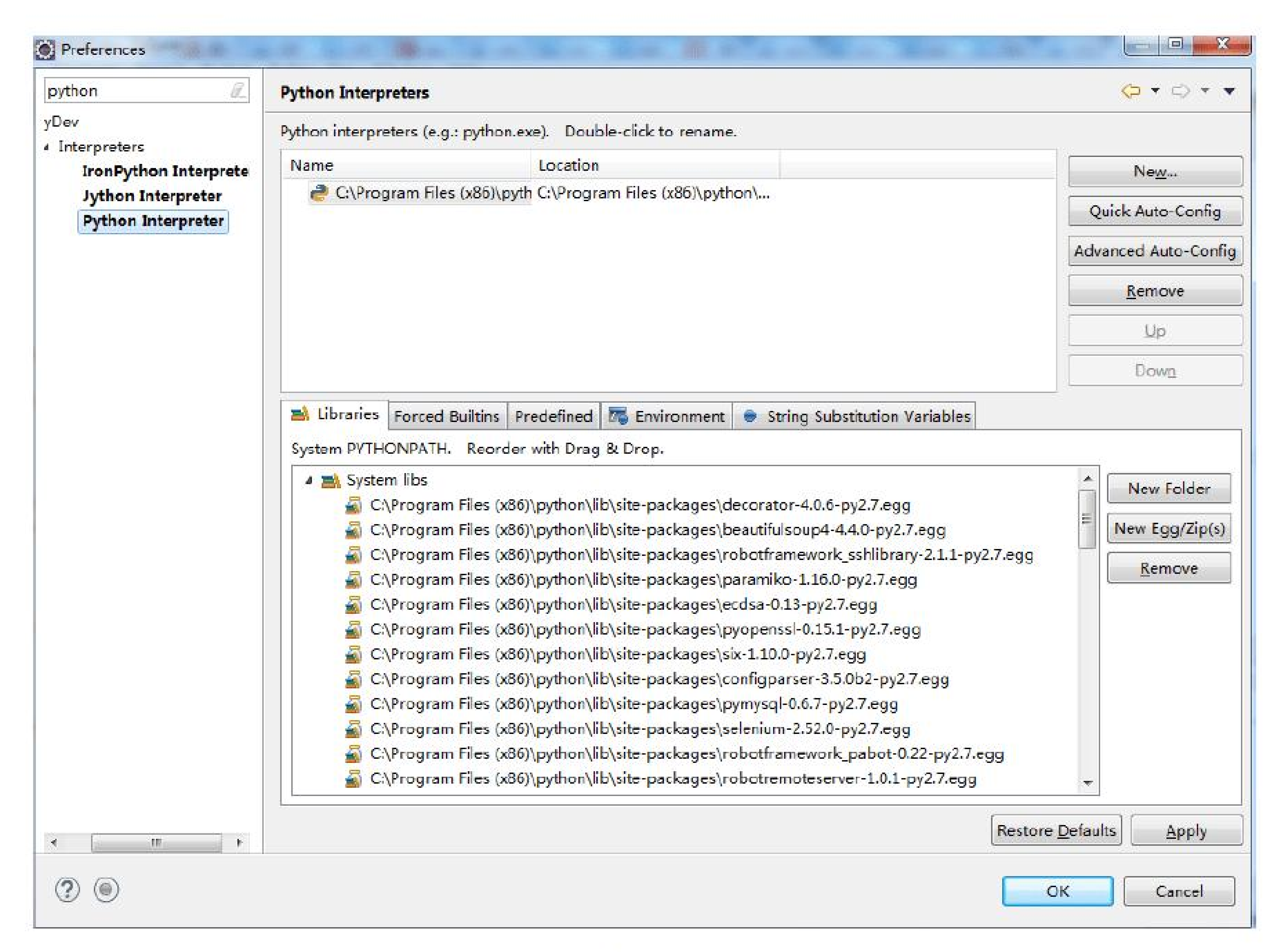


图 6-1-8

插件配置完成后，我们就可以使用 Eclipse 来构建一个 Python 项目了。这里我们新建一个 ExcelLibrary 项目，在 Project name 文本框中输入“ExcelLibrary”，然后单击 Finish 按钮完成项目创建，如图 6-1-9 所示。

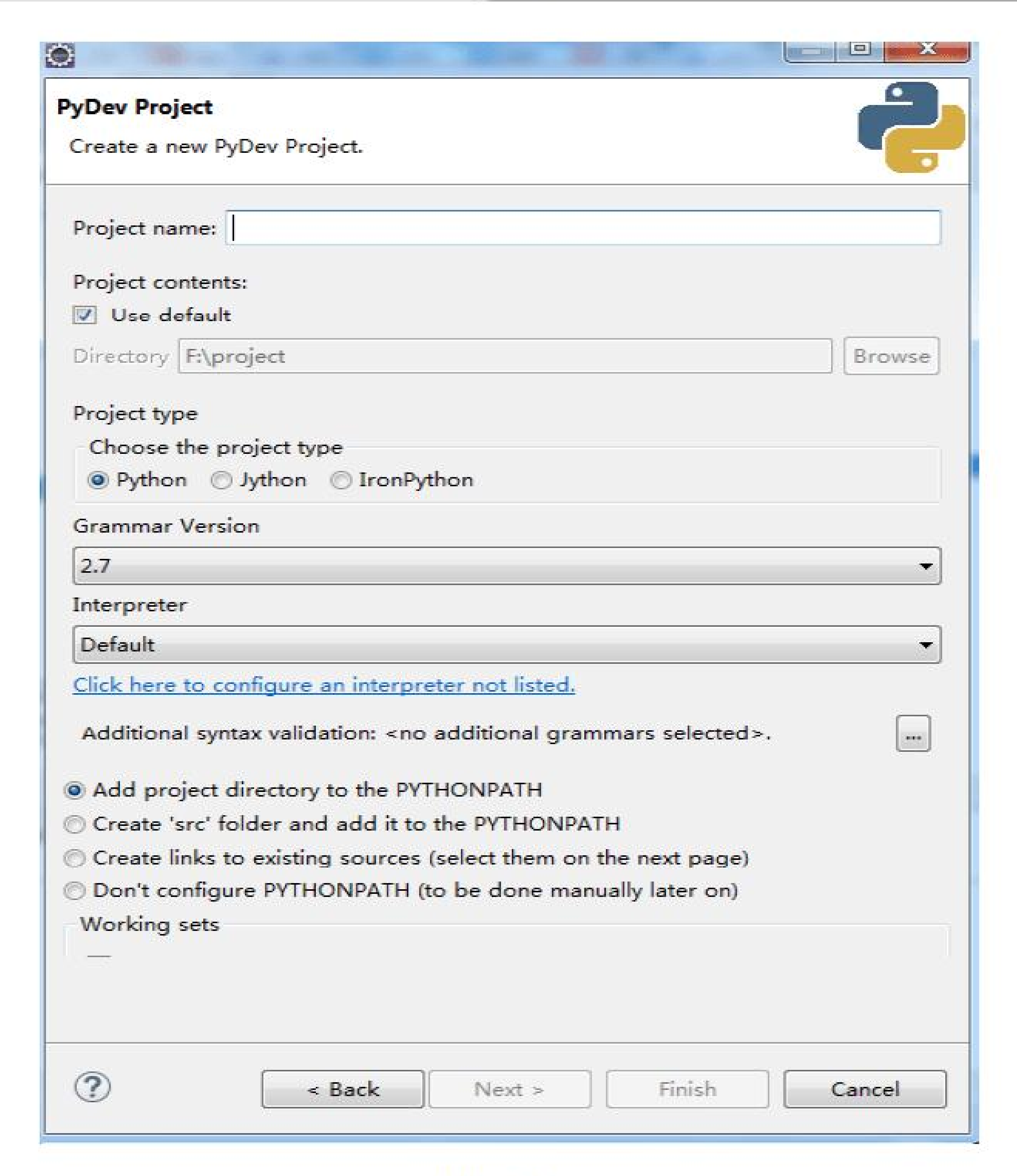


图 6-1-9

6.1.2　使用 Python 编写自定义的 Lib

Lib 项目创建完成后，我们就可以编写自己的 Lib 了。这里我们编写一个从 Excel 文件中读取数据的 Lib 示例。

```
# -*- coding: utf-8 -*-
'''
导入操作 excel 需要第三方的 xlrd Library
'''
import xlrd
'''
引入 robotFramework 的日志输出 logger
'''
from robot.api import logger
'''
定义一个 python class
'''
class ExcelUtil():
    def __init__(self):
        return
    '''
        打开一个 excel 文件
    '''
```

```
    def open_excel(self,excelfile):
        try:
            data = xlrd.open_workbook(excelfile)
            return data
        except Exception,e:
            logger.error(e)
    '''
    获取 excel 中的数据方法，通过参数指定需要读取的 excel 文件名和 sheetname
    '''
    def get_excel_bysheetname(self,excelfile,lineindex=0,sheetname='Sheet1'):
        data = self.open_excel(excelfile)
        sheet = data.sheet_by_name(sheetname)
        rows = sheet.nrows
        linedata = sheet.row_values(lineindex)
        list = []
        for rownum in range(1,rows):
            row = sheet.row_values(rownum)
            if row:
                app = {}
                for j in range(len(linedata)):
                    app[linedata[j]] = row[j]
                list.append(app)
        return list
```

在示例代码中，定义了函数 get_excel_bysheetname 来获取 Excel 文件中的数据，可以通过参数来指定需要获取 Excel 文件哪个 sheet 中的数据，获取到的 sheet 数据最终以 List 的形式返回。List 中的每一条记录都是以 Python 中的字典形式存储进去的。

我们可以调用一下我们写的 lib，看看是否可以正常使用。在 RIDE 中，我们导入刚刚写的 Lib，如图 6-1-10 所示。

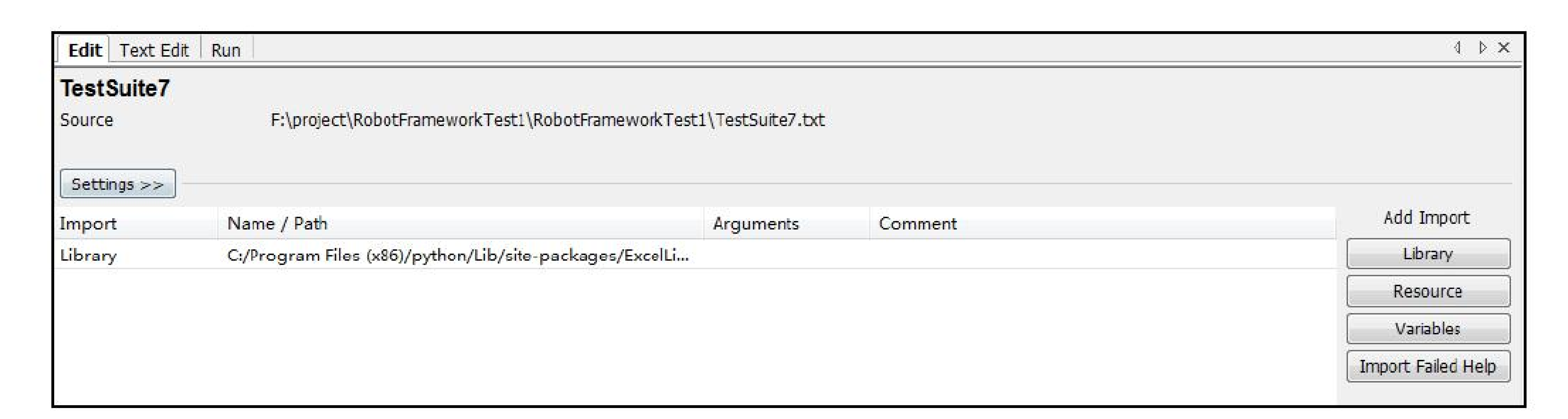


图 6-1-10

然后在 RIDE 中按 F5 快捷键，可以看到我们自定义的关键字，如图 6-1-11 所示。

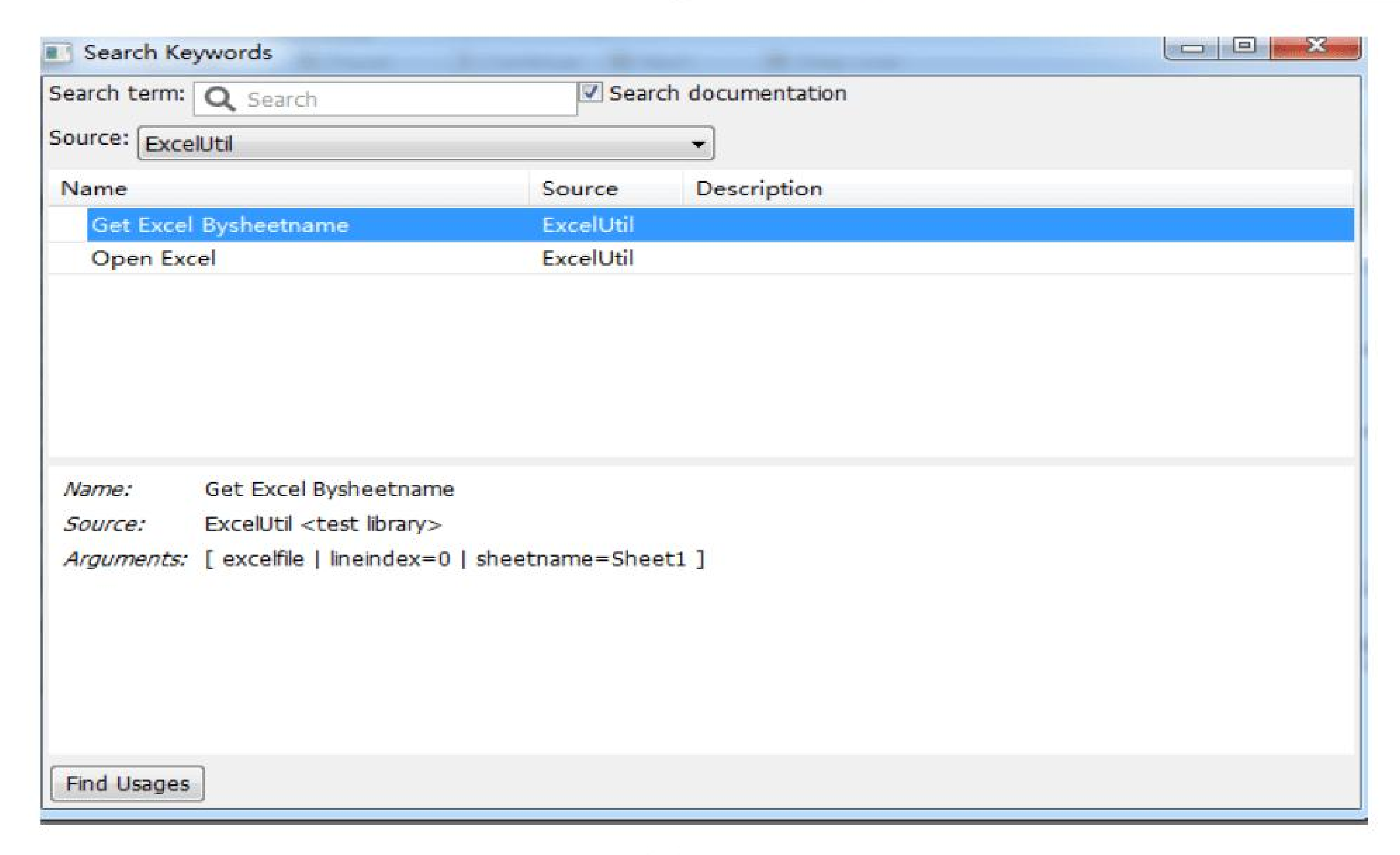


图 6-1-11

导入后，我们可以通过一个测试用例调用一下，并且将结果以 log 形式输出。

```
${list} Get Excel Bysheetname    E:\\task.xls
log ${list}
```

运行结果如图 6-1-12 所示。

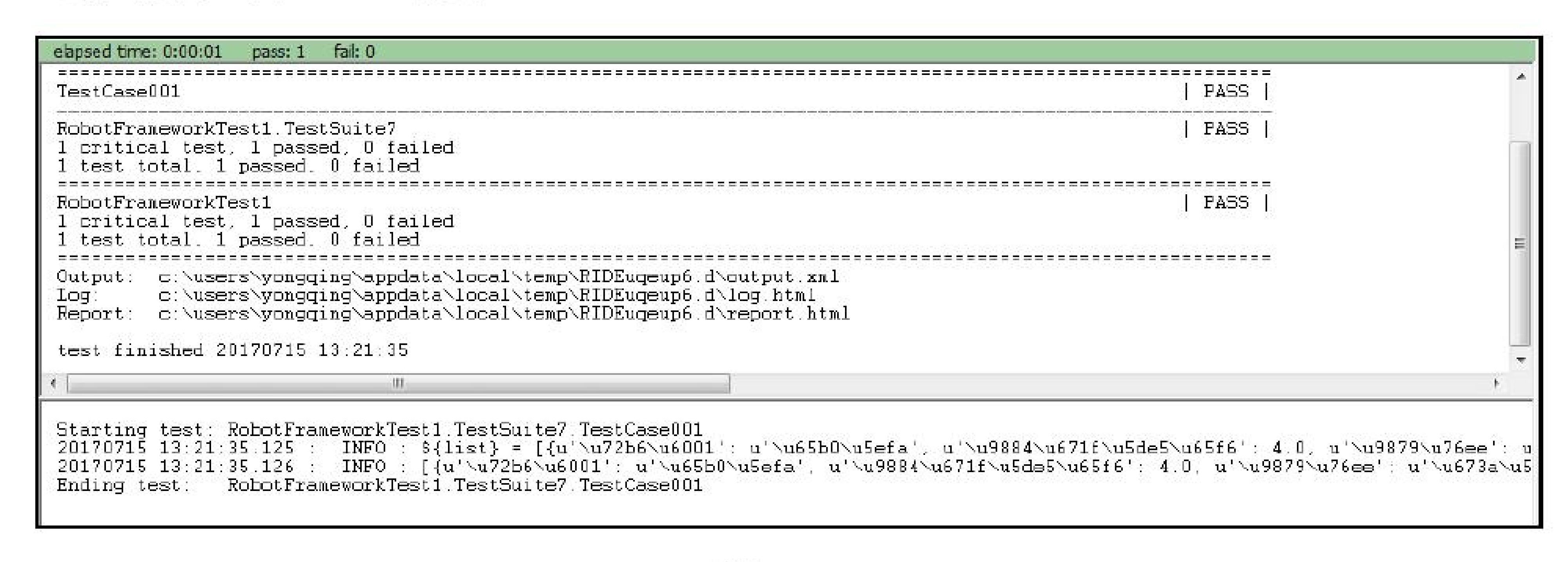


图 6-1-12

从结果我们可以看到，以 Unicode 格式的形式输入了一个 List，List 中的每一个元素都是一个字典。

6.1.3　打包自定义的 Lib

Python 打包自定义的 Lib 需要遵循 Python 语言的模块导入规范，我们需要首先建立一个 __init__.py 文件，如图 6-1-13 所示。



图 6-1-13

然后在__init__.py 中定义需要对外提供的 Library，在 import 中需要导入已经编写好的自定义的 Library 模块，并且导入已经定义好的 Library 的版本文件，如图 6-1-14 和图 6-1-15 所示。



图 6-1-14

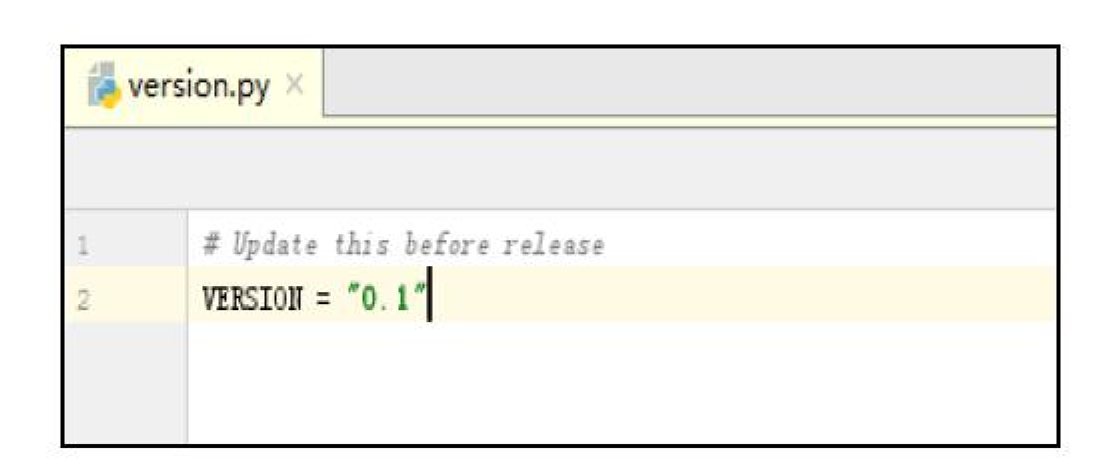


图 6-1-15

定义好的 Library 的 package 层次如图 6-1-16 所示。

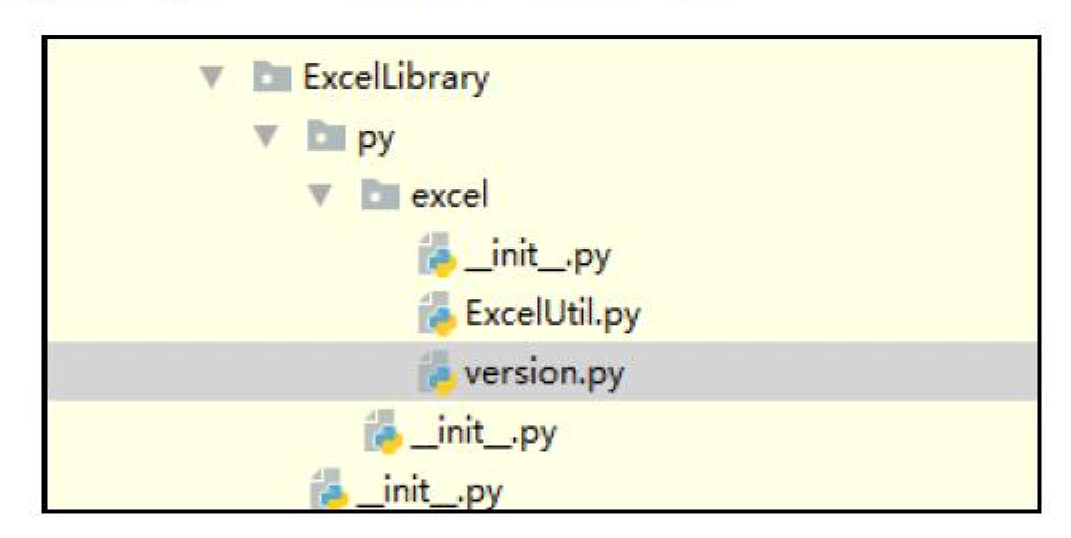


图 6-1-16

上面这些全部完工后，我们就可以把做好的 Library 放到 Python 的 Lib\site-packages 目录下了。这样在 RIDE 中就可以直接导入我们定义的名称为 ExcelLibrary 的 Library 了，如图 6-1-17 所示；并且通过 F5 快捷键也可以看到 Library 下的关键字，如图 6-1-18 所示。

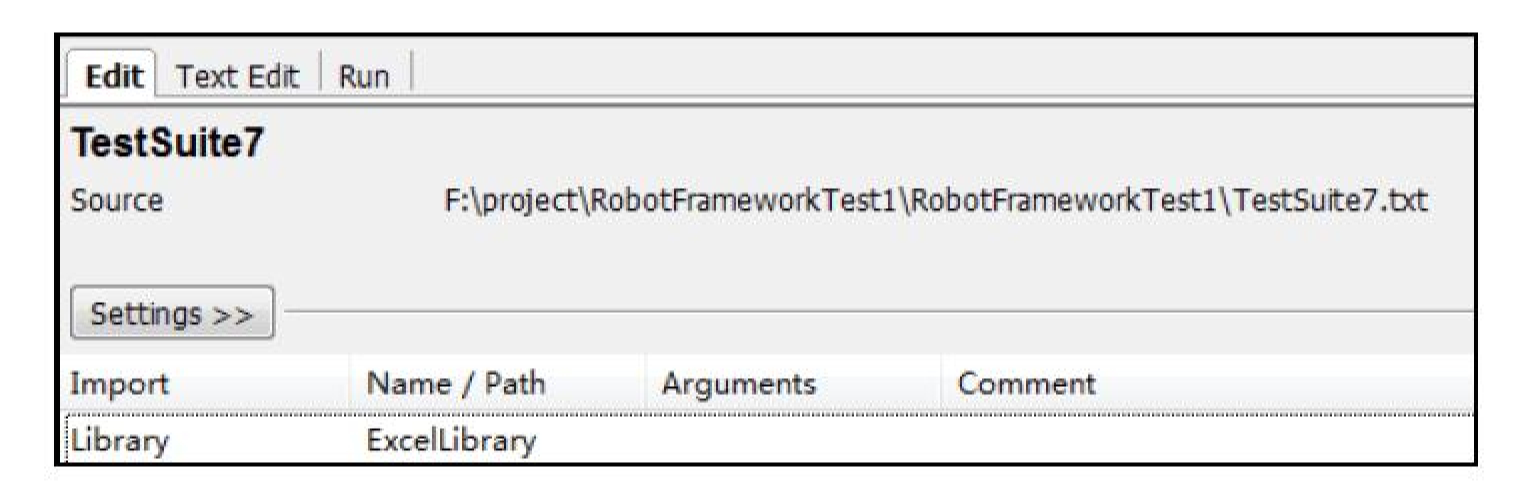


图 6-1-17

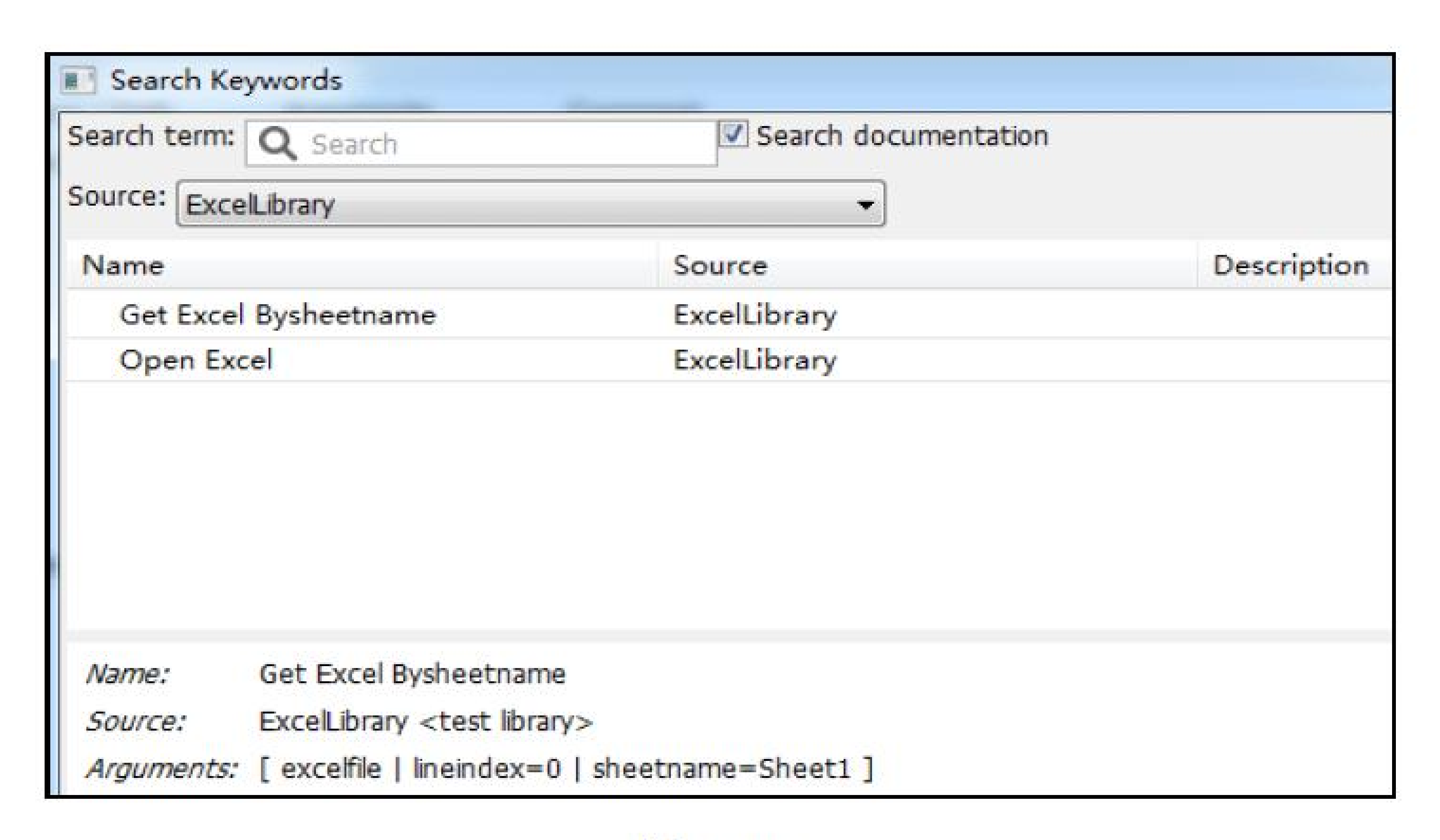


图 6-1-18

6.1.4　Remote 远程库

在自动化测试时，有时并不是所有的 Library 库都安装部署在本地运行环境上，或者说为了解决本地自动化测试环境安装多个 Library 库所带来的复杂工作量，Robot Framework 设计了通过 Remote 的方式来进行远程调用。通过远程调用的方式，可以调用服务器上的远程库，调用时采用 XML-RPC 远程调用协议。该方式还可以解决跨语言调用的问题。因为采用了远程调用的方式，所以服务端的远程库可以是多种不同的语言，只需要提供远程调用服务即可。图 6-1-19 中的流程描述了一个测试用例在执行时如何通过 XML-RPC 的形式来调用远程库的过程。XML-RPC 协议本质是一个 HTTP 协议通信下的远程调用方式。

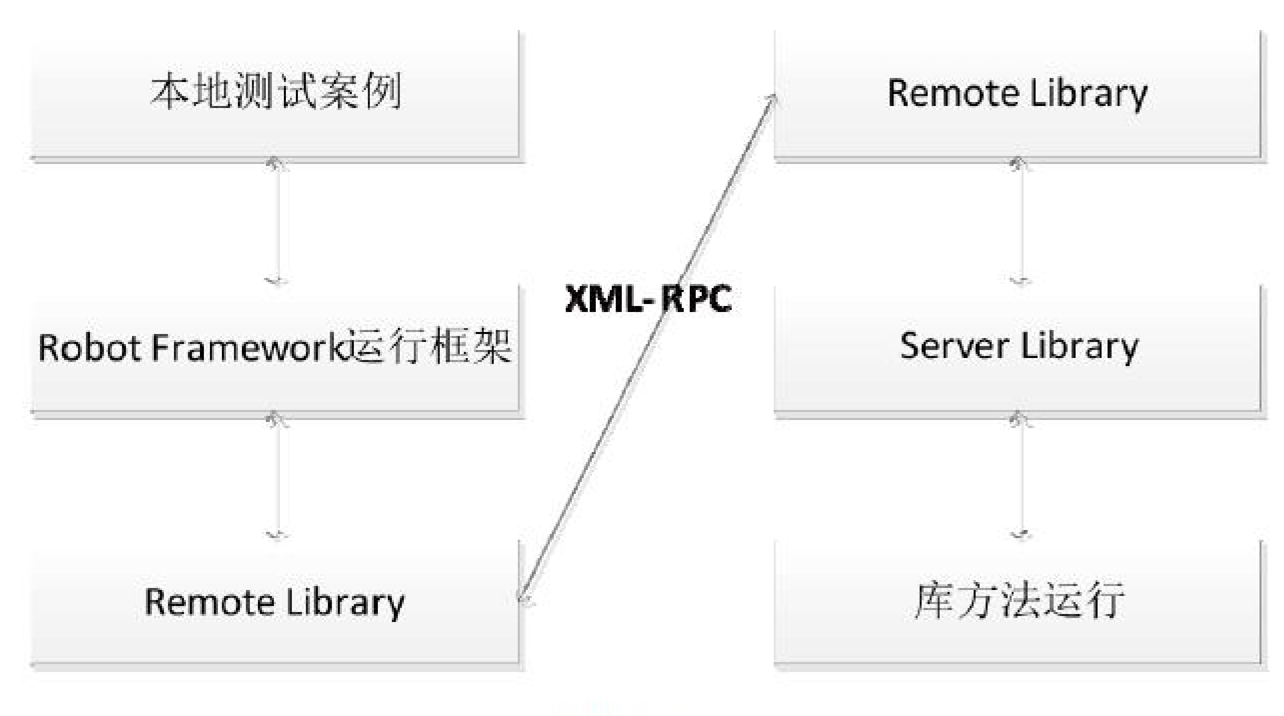


图 6-1-19

Robot Framework 提供了远程调用接口库，可以通过 GitHub（地址为 https://github.com/robotframework/RemoteInterface）来获取远程 Remote 接口库。Robot Framework 中提供了 8 种不同语言的版本，囊括了 Python、Java、Ruby、.net、Clojure、Perl、Node.js 和 PHP。Robot Framework 正是通过远程调用接口库来实现跨语言支持的。这些不同的语言版本，都可以在 GitHub 中下载到。我们在后面讲到的使用 Java 来编写自定义的 Lib 库也是会基于远程调用接口协议来进行实现的。

这里我们以 Python 库为例，讲述 Remote 库的使用过程。远程服务配置如表 6-1-1 所示。

表 6-1-1　远程服务配置信息

服务端参数	默认值	说明
host	'127.0.0.1'	服务端的启动时的服务监听地址，一般建议使用‘0.0.0.0’来启动服务端，这样可以保证外部的客户端都可以调用和访问
port	8270	服务端监听服务启动时指定的启动端口号，默认为 8270。端口号如果被占用，可以根据实际情况进行调整和修改
port_file	None	将启动占用的 port 输出到一个文件中
allow_remote_stop	True	这是一个启动参数，默认值为 True，表示允许在调用远程 Server 接口时，可以通过关键字 Stop Remote Server 来停止远程接口服务

首先需要自己定义一个远程 Python 接口服务端，然后将其启动起来。想要自己定义一个远程调用服务端，需要通过 from robotremoteserver import RobotRemoteServer 引入 RobotRemoteServer 服务端。然后调用 RobotRemoteServer 服务的初始化方法。在如下的 Python 脚本示例中，我们自己定义了启动远程调用监听服务的地址和端口。

```
RemoteExample.py
#coding:utf8
from robotide.lib.robot.libraries.String import String
from robotremoteserver import RobotRemoteServer
class RemoteExample:
   def __int__(self):
      pass
if __name__ == '__main__':
   RobotRemoteServer(String(), host='0.0.0.0',
port=8270,port_file='d://remote-port.txt')
```

然后运行 RemoteExample.py 脚本，会在本地启动我们需要的远程接口服务，如图 6-1-20 所示。

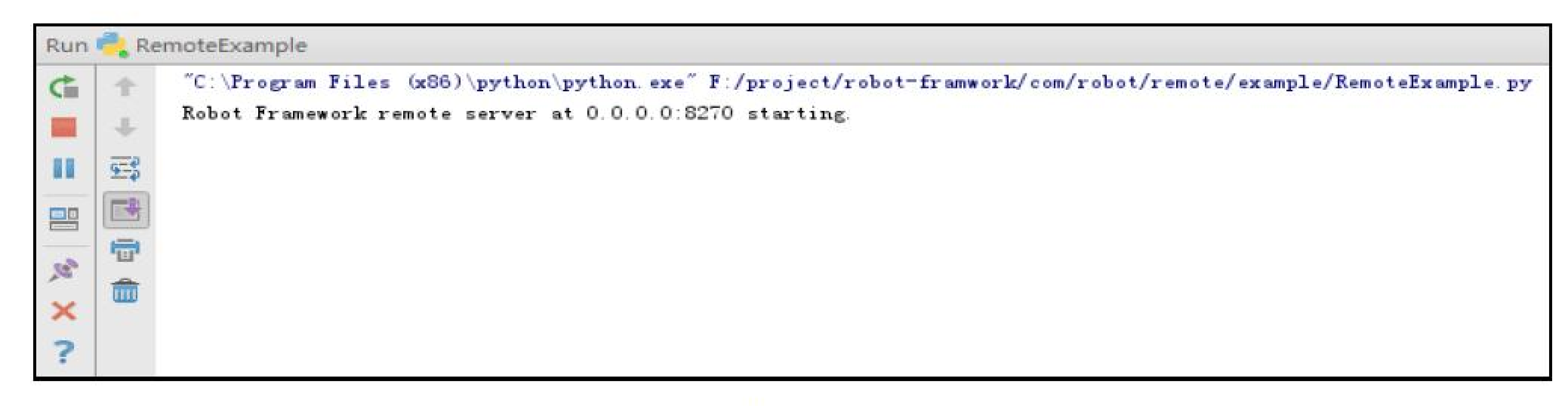


图 6-1-20

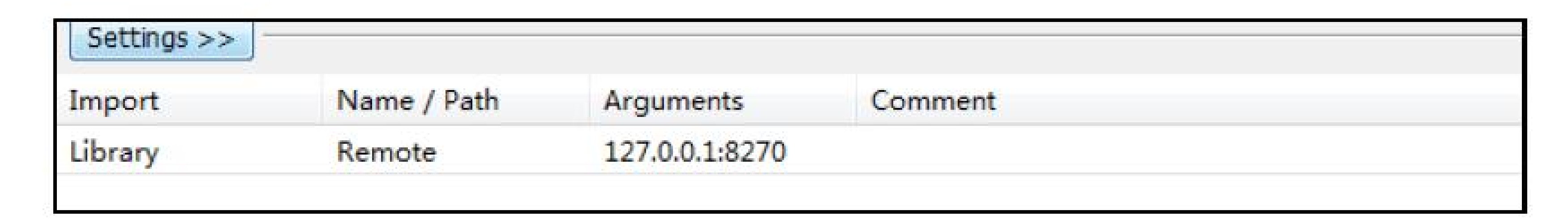
我们再来简单分析一下 robotremoteserver.py 脚本。在脚本的初始化方法中，我们也可以看到需要传入的参数，包括 library、host、port、port_file、allow_stop，library 就是远程接口服务启动时需要将服务端的本地库加载为远程调用接口服务的 library。在上面我们启动的自定义服务中，library 加载的是 robotide.lib.robot.libraries.String 中的 String 库。host 和 port 如果不传入的话，默认为 127.0.0.1 和 8270，和我们上面提供的表格中的远程服务配置参数是一致的，如图 6-1-21 所示。

```
def __init__(self, library, host='127.0.0.1', port=8270, port_file=None,
             allow_stop=True):
    """Configure and start-up remote server.

    :param library:     Test library instance or module to host.
    :param host:        Address to listen. Use ``'0.0.0.0'`` to listen
                        to all available interfaces.
    :param port:        Port to listen. Use ``0`` to select a free port
                        automatically. Can be given as an integer or as
                        a string.
    :param port_file:   File to write port that is used. ``None`` means
                        no such file is written.
    :param allow_stop:  Allow/disallow stopping the server using
                        ``Stop Remote Server`` keyword.
    """
    SimpleXMLRPCServer.__init__(self, (host, int(port)), logRequests=False)
    self._library = library
    self._allow_stop = allow_stop
    self._shutdown = False
    self._register_functions()
    self._register_signal_handlers()
    self._announce_start(port_file)
    self.serve_forever()
```

图 6-1-21

自定义的远程调用接口服务启动后，我们在 RIDE 客户端中可以通过 import 的方式导入远程调用接口库，如图 6-1-22 所示。


Settings >>

Import	Name / Path	Arguments	Comment
Library	Remote	127.0.0.1:8270	

图 6-1-22

此时我们通过 F5 快捷键查看一下 Remote 库下的关键字方法，如图 6-1-23 所示。

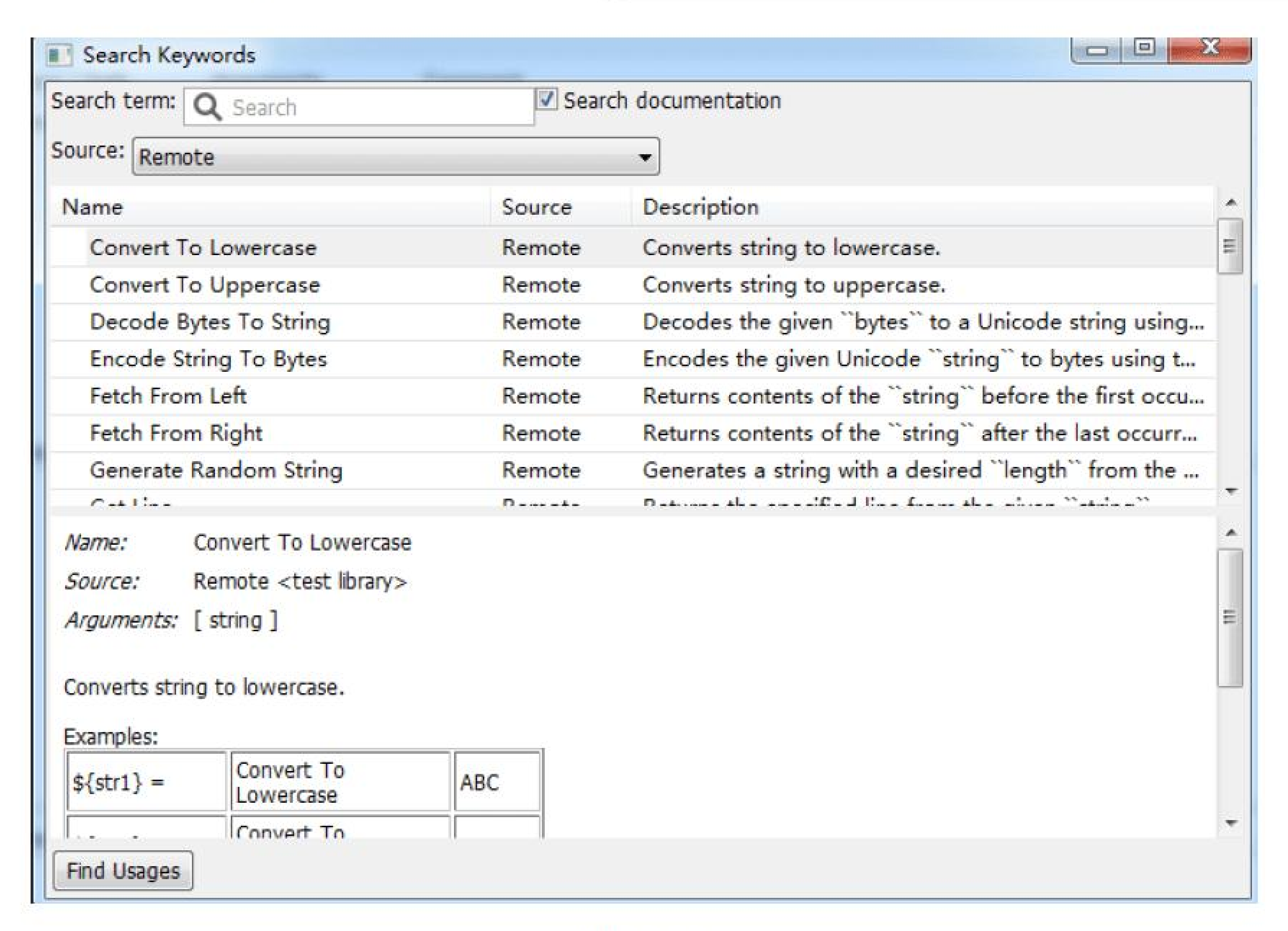
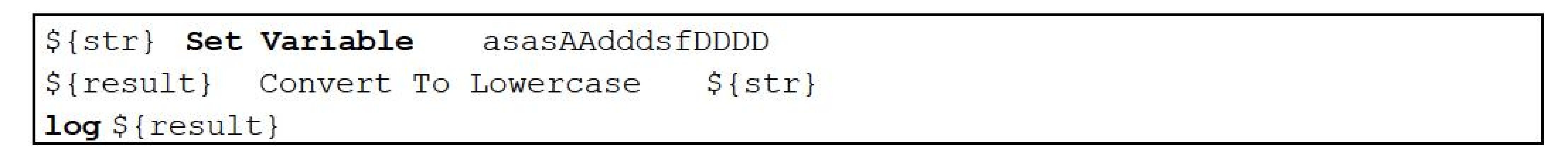


图 6-1-23

可以看到 Remote 下包括 String Library 库下的所有关键字，服务端本地的 String Library 库就变成了一个远程接口服务以供我们调用。

【示例】调用 Remote 下的 Convert To Lowercase 关键字。



```
${str}  Set Variable    asasAAdddsfDDDD
${result}  Convert To Lowercase    ${str}
log ${result}
```

运行结果如图 6-1-24 所示。



图 6-1-24

在实际使用时，建议你通过 netsh 工具抓包的方式去查看 Remote 远程调用接口服务在调用时是如何进行通信的。

在自定义远程接口服务中，除了可以使用 robotide.lib.robot.libraries 中已有的 Library 外，也可以使用我们自己编写的 Library。下面我们自定义一个 Library：

```
ExampleLibrary.py
#coding:utf8
class ExampleLibrary(object):
   def __int__(self):
      pass
   def add(self,x,y):
      print 'serivce is called'
      return int(x)+int(y)
   pass
```

定义好后，在 RobotRemoteServer 中使用 ExampleLibrary 来进行启动，如图 6-1-25 所示。

```
#coding:utf8
from robotide.lib.robot.libraries.String import String
from robotremoteserver import RobotRemoteServer
from ExampleLibrary import ExampleLibrary
class RemoteExample:
   def __int__(self):
      pass
if __name__ == '__main__':
   RobotRemoteServer(ExampleLibrary(), host='0.0.0.0',
port=8270,port_file='d://remote-port.txt')
```

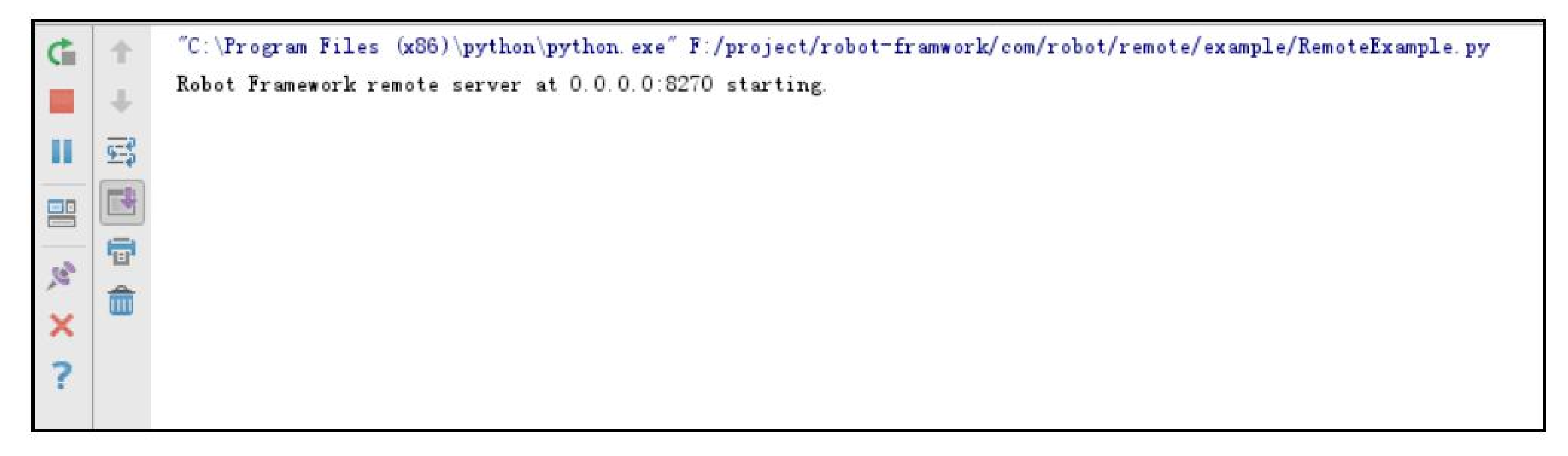


图 6-1-25

启动成功后，我们可以通过 F5 快捷键来查看自定义的 ExampleLibrary 库中的关键字，如图 6-1-26 所示。

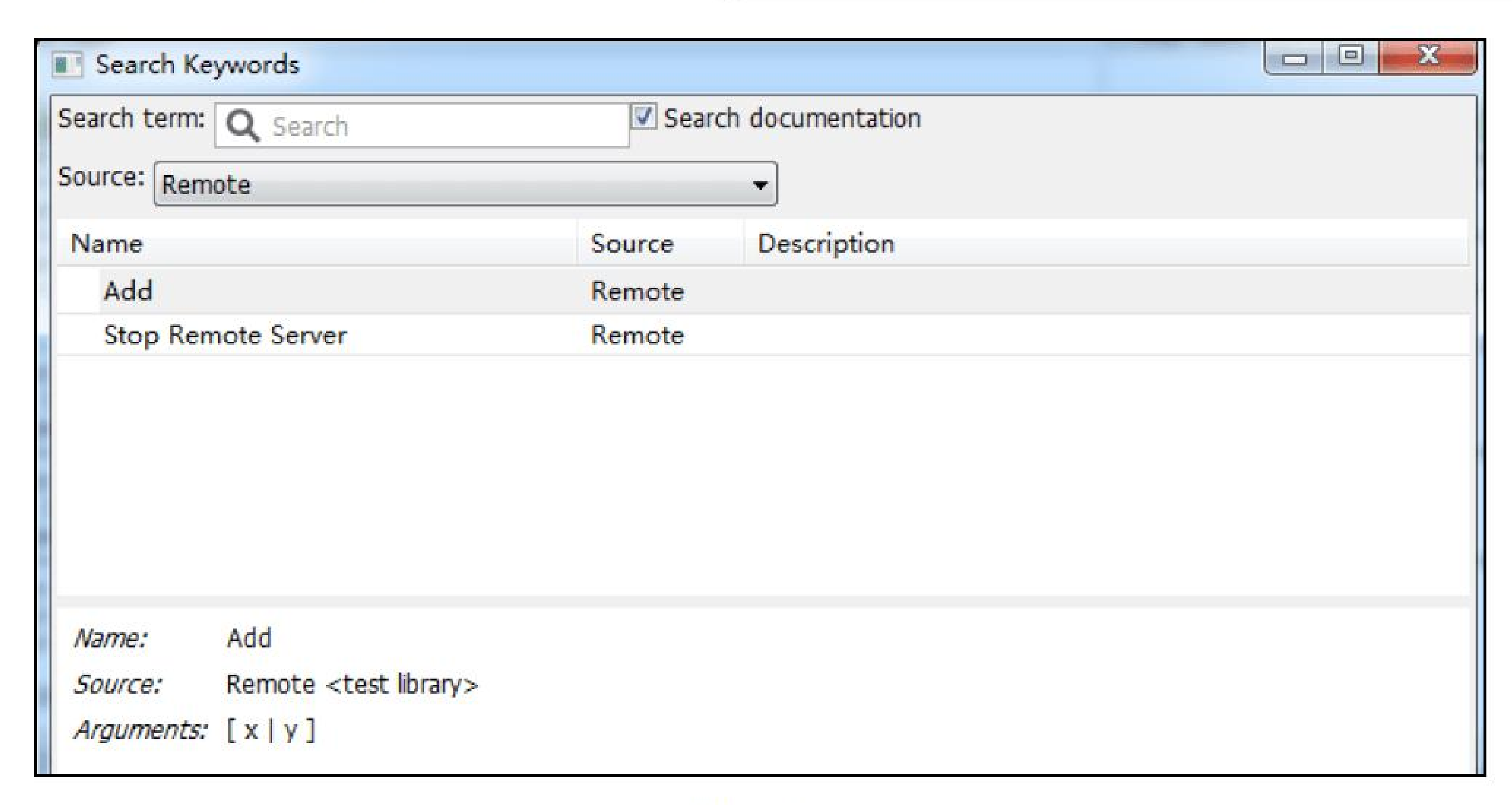


图 6-1-26

调用一下我们自定义的 ExampleLibrary 库中的 Add 关键字。

```
${result}  Add  3   4
log ${result}
```

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite8.TestCase0002
20180812 17:07:33.531 :  INFO : serivce is called
20180812 17:07:33.532 :  INFO : ${result} = 7
20180812 17:07:33.533 :  INFO : 7
Ending test:  RobotFrameworkTest1.TestSuite8.TestCase0002
```

6.2 使用 Java 编写自定义的 Robot Framework Lib

6.2.1 在 Robot Framwork 中调用 Java Lib 库

我们在前面介绍了 Robot Framework 可以支持跨语言，对 Java 也是可以支持的。在 Robot Framework 中，RIDE 本身提供了对测试用例的两种执行方式，支持 pybot 和 jybot 两种执行方式，如图 6-2-1 所示。

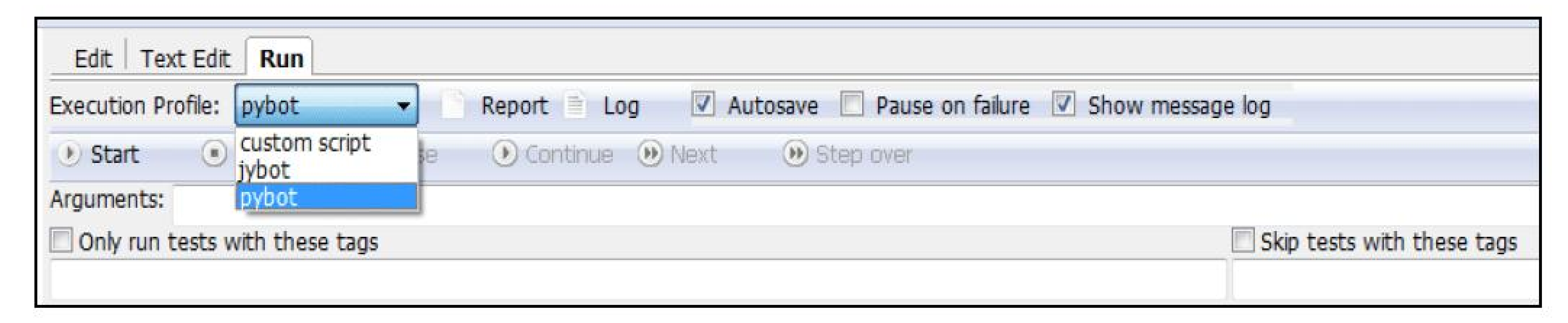


图 6-2-1

jybot 就是以 Java 虚拟机的形式来执行测试用例，在执行的环境中需要安装 Java JDK 运行环境以及 Jython 安装包。Jython 是一个 Python 语言在 Java 中的完全实现，也就是说用 Java 来实现了 Python 语言的功能。Jython 中既提供了 Python 库，也支持 Java 方法的执行，解决了 Robot Framework 中 Python 不支持直接调用 Java 语言的问题。Jython 可以从 http://www.jython.org/downloads.html 页面中进行下载，安装完成后就可以在 Robot Framework 中使用了，如图 6-2-2 所示。

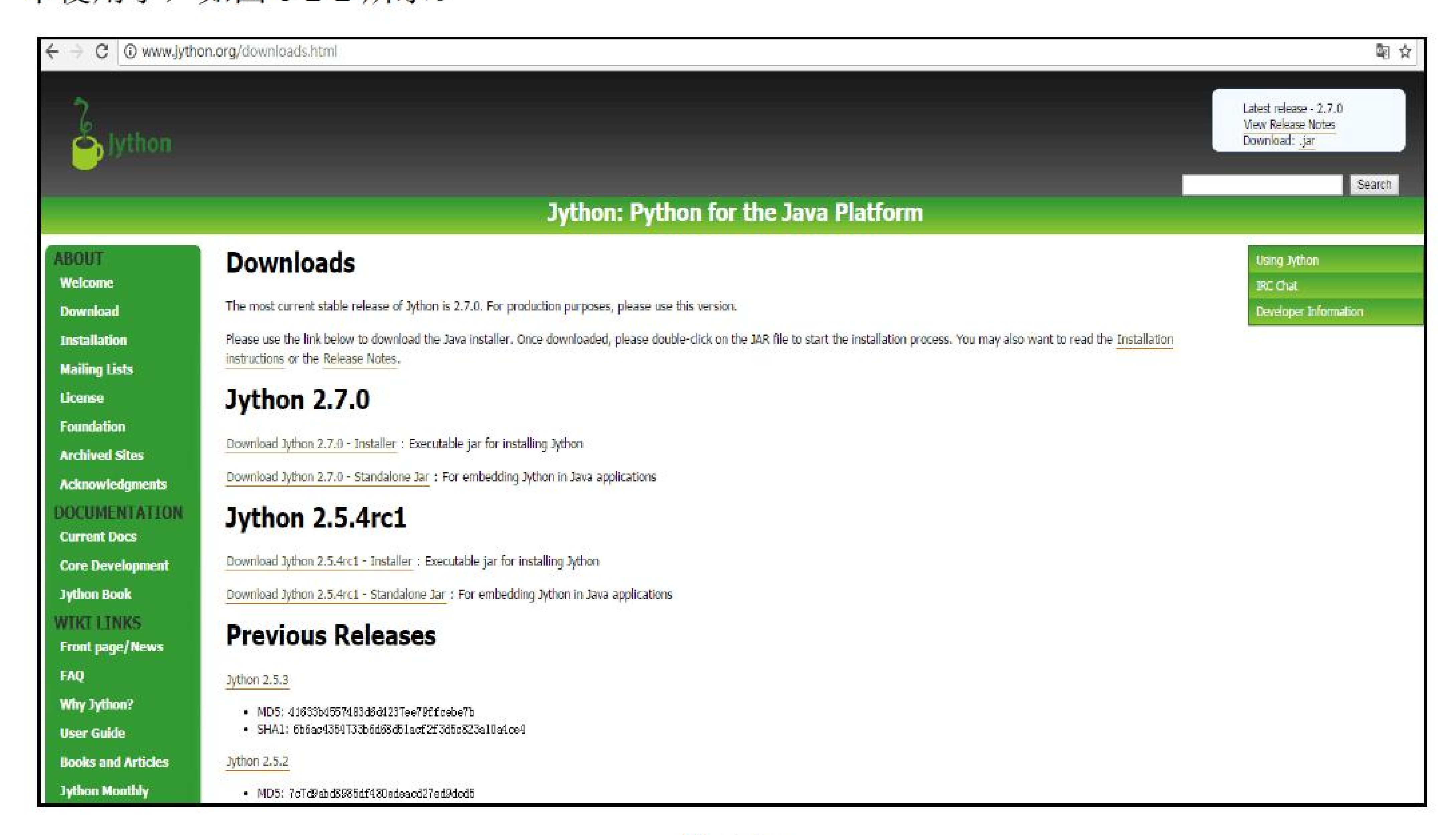
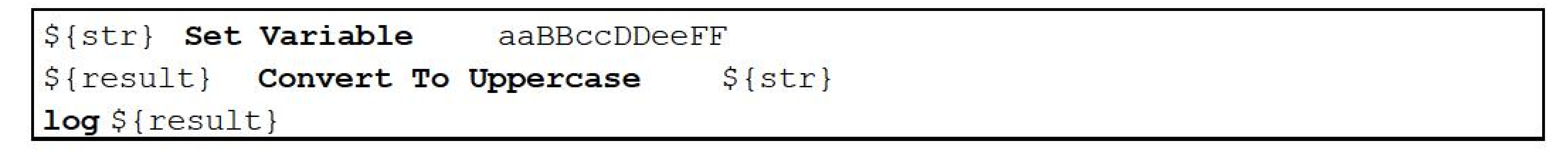


图 6-2-2

下面我们来看一个 Jython 语言执行测试用例的例子。

【示例】用 Jython 语言将字符串全部转换成大写形式。



```
${str}  Set Variable    aaBBccDDeeFF
${result}  Convert To Uppercase    ${str}
log ${result}
```

运行结果如图 6-2-3 所示。这里我们选择使用 jybot 来执行，同 pybot 一样得到了我们想要的运行结果。

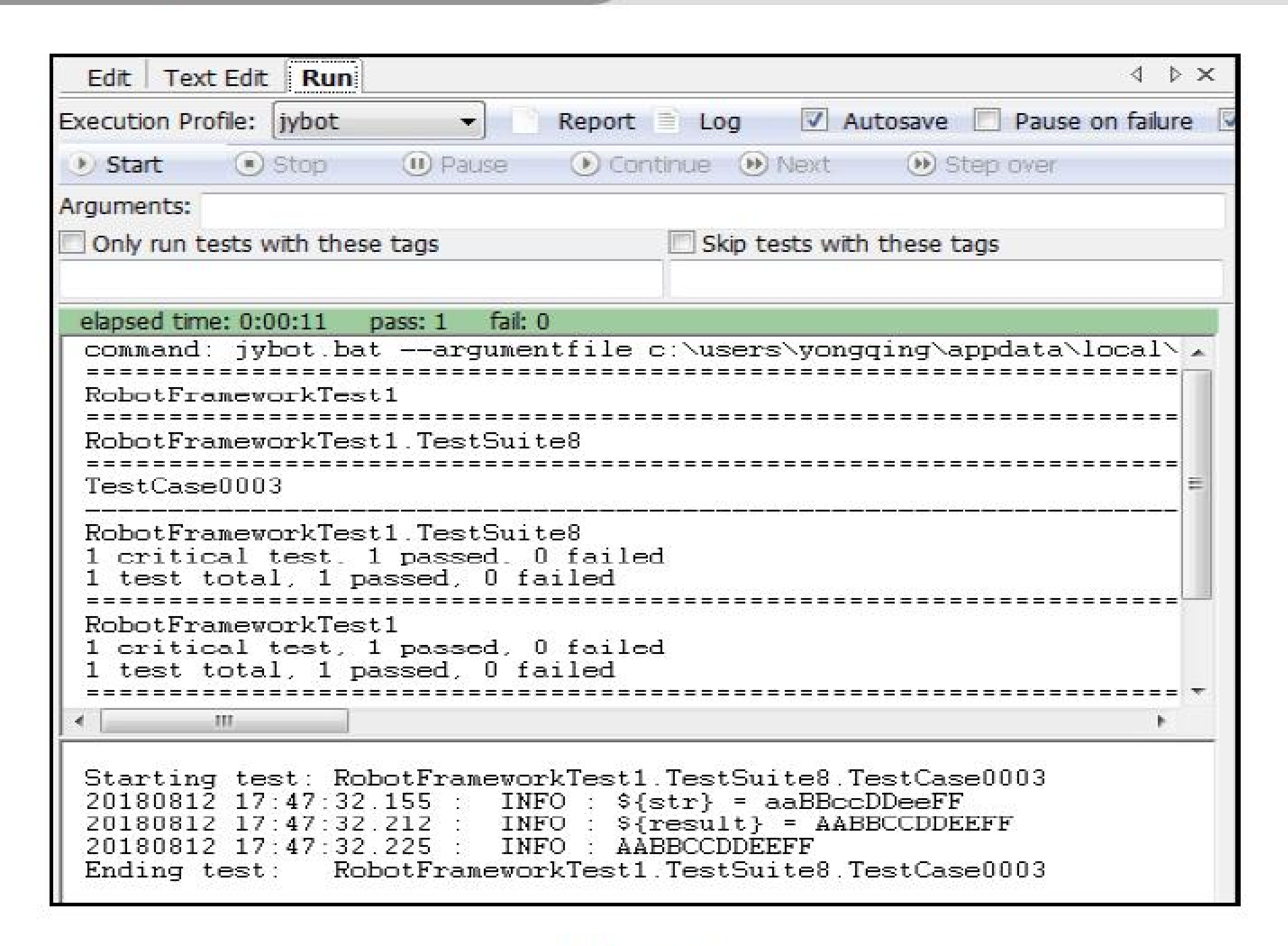


图 6-2-3

其实在 Robot Framework 的官方网站中很多 Library 库都提供了 Java 语言版本的实现，如图 6-2-4 所示。

STANDARD	EXTERNAL	OTHER
Android library Library for all your Android automation needs. It uses Calabash Android internally.	AnywhereLibrary Library for testing Single-Page Apps (SPA). Uses Selenium Webdriver and Appium internally.	AppiumLibrary Library for Android- and iOS-testing. It uses Appium internally.
Archive library Library for handling zip- and tar-archives.	AutoItLibrary Windows GUI testing library that uses AutoIt freeware tool as a driver.	CncLibrary Library for driving a CNC milling machine.
Database Library (Java) Java-based library for database testing. Usable with Jython. Available also at Maven central.	Database Library (Python) Python based library for database testing. Works with any Python interpreter, including Jython.	Diff Library Library to diff two files together.
Django Library Library for Django, a Python web framework.	Eclipse Library Library for testing Eclipse RCP applications using SWT widgets.	robotframework-faker Library for Faker, a fake test data generator.
FTP library Library for testing and using FTP server with Robot Framework.	HTTP library (livetest) Library for HTTP level testing using livetest tool internally.	HTTP library (Requests) Library for HTTP level testing using Request internally.
HttpRequestLibrary (Java) Library for HTTP level testing using Apache HTTP client. Available also at Maven central.	iOS library Library for all your iOS automation needs. It uses Calabash iOS Server internally.	ImageHorizonLibrary Cross-platform, pure Python library for GUI automation based on image recognition.
JavaFXLibrary Library for testing JavaFX applications, based on TestFX. Has also remote interface support.	MongoDB library Library for interacting with MongoDB using pymongo.	MQTT library Library for testing MQTT brokers and applications.
NcclientLibrary https://github.com/ncclient/ncclient	Rammbock Generic network protocol test library that offers easy way to specify network packets and inspect the results of sent and received packets.	RemoteSwingLibrary Library for testing and connecting to a java process and using SwingLibrary, especially Java Web Start applications.

图 6-2-4

这里以我们在第 2 章中讲到的 Database Library Python 库作为示例，看一下这个库对应的 Java 语言的实现以及如何通过 Java 语言的形式来进行调用。

在使用时，可以通过 Robot Framework 提供的 maven 插件 robotframework-maven-plugin 来直接引入：

```
<dependency>
    <groupId>com.github.hi-fi</groupId>
    <artifactId>robotframework-dblibrary</artifactId>
    <version>3.1.1</version>
</dependency>
```

或者将源码下载下来，通过执行编译打包的方式生成。这里我们选择直接用源码的形式进行编译，源码可以从 https://github.com/Hi-Fi/robotframework-dblibrary 上直接获取到。下载完成，解压到自己需要的目录后，可以通过 cmd 命令行切入对应的目录下，使用 maven 命令行进行编译，前提是需要在自己的环境中事先安装好 maven 的编译环境。编译打包时执行 mvn clean install -Dmaven.test.skip=true 即可生成我们想要的 Java 语言实现的 Library 库，如图 6-2-5 和图 6-2-6 所示。

```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\yongqing>f:

F:\>cd F:\project\remotejava\jrobotremoteserver-master\robotframework-dblibrary-
master\robotframework-dblibrary-master

F:\project\remotejava\jrobotremoteserver-master\robotframework-dblibrary-master\
robotframework-dblibrary-master>mvn clean install -Dmaven.test.skip=true
[WARNING]
[WARNING] Some problems were encountered while building the effective settings
[WARNING] Unrecognised tag: 'id' (position: START_TAG seen ...</mirror>\r\n
-->\r\n\t<id>... @160:6)  @ C:\apache-maven-3.3.3-bin\apache-maven-3.3.3\conf\se
ttings.xml, line 160, column 6
[WARNING]
[INFO] Scanning for projects...
Downloading: https://repo.maven.apache.org/maven2/org/sonatype/plugins/nexus-sta
ging-maven-plugin/1.6.3/nexus-staging-maven-plugin-1.6.3.pom
Downloaded: https://repo.maven.apache.org/maven2/org/sonatype/plugins/nexus-stag
ing-maven-plugin/1.6.3/nexus-staging-maven-plugin-1.6.3.pom (12 KB at 6.9 KB/sec
)
Downloading: https://repo.maven.apache.org/maven2/org/sonatype/nexus/maven/nexus
-staging/1.6.3/nexus-staging-1.6.3.pom
Downloaded: https://repo.maven.apache.org/maven2/org/sonatype/nexus/maven/nexus-
```

图 6-2-5

```
管理员: C:\Windows\system32\cmd.exe
y-3.2-SNAPSHOT.jar to C:\apache-maven-3.3.3-bin\apache-maven-3.3.3\repo\com\gith
ub\hi-fi\robotframework-dblibrary\3.2-SNAPSHOT\robotframework-dblibrary-3.2-SNAP
SHOT.jar
[INFO] Installing F:\project\remotejava\jrobotremoteserver-master\robotframework
-dblibrary-master\robotframework-dblibrary-master\pom.xml to C:\apache-maven-3.3
.3-bin\apache-maven-3.3.3\repo\com\github\hi-fi\robotframework-dblibrary\3.2-SNA
PSHOT\robotframework-dblibrary-3.2-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:33 min
[INFO] Finished at: 2018-08-12T18:11:52+08:00
[INFO] Final Memory: 22M/180M
[INFO] ------------------------------------------------------------------------

F:\project\remotejava\jrobotremoteserver-master\robotframework-dblibrary-master\
robotframework-dblibrary-master>
```

图 6-2-6

打包完成后，生成的 target 目录中就可以获取到编译好的 Java 语言实现的 Library，如图 6-2-7 所示。

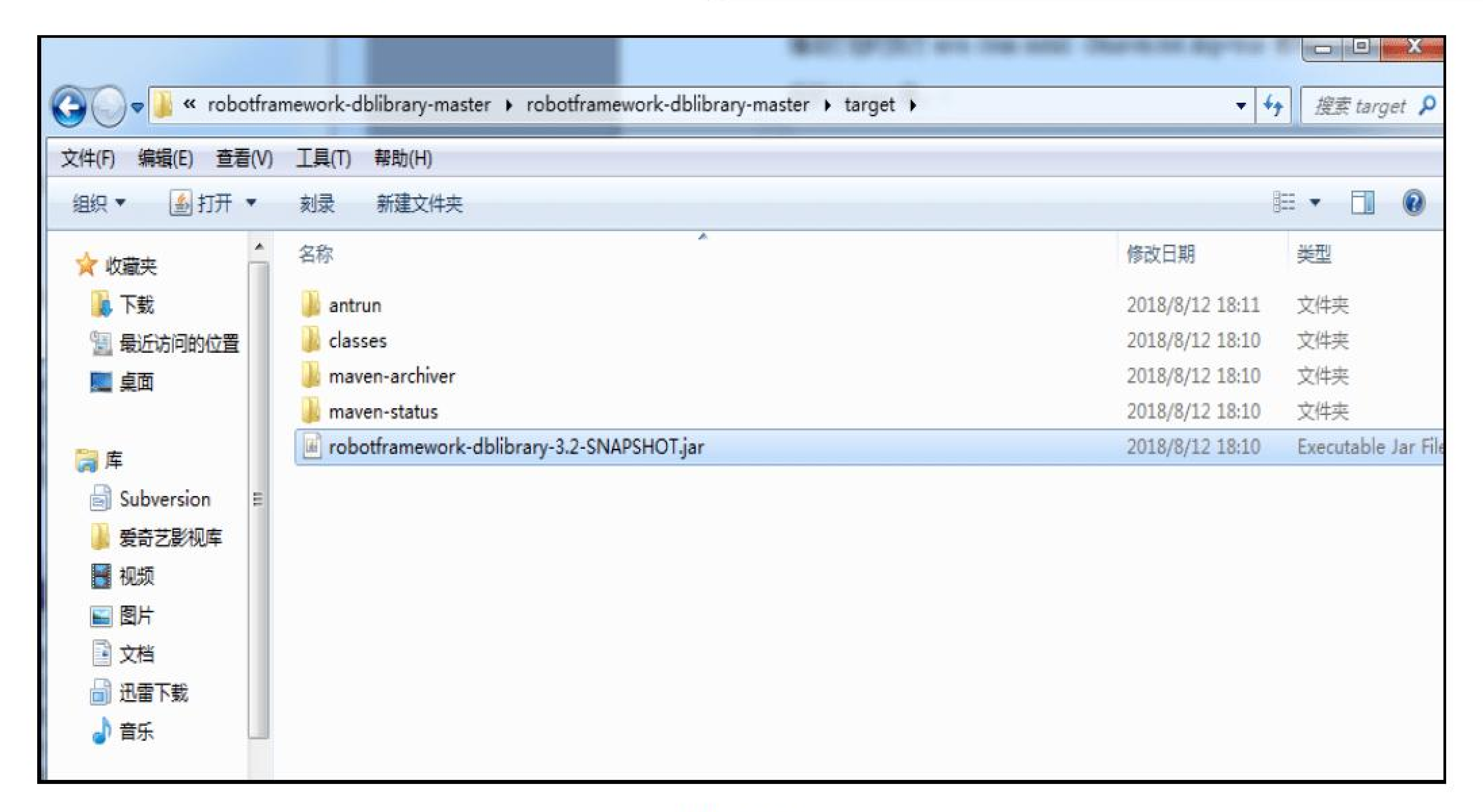
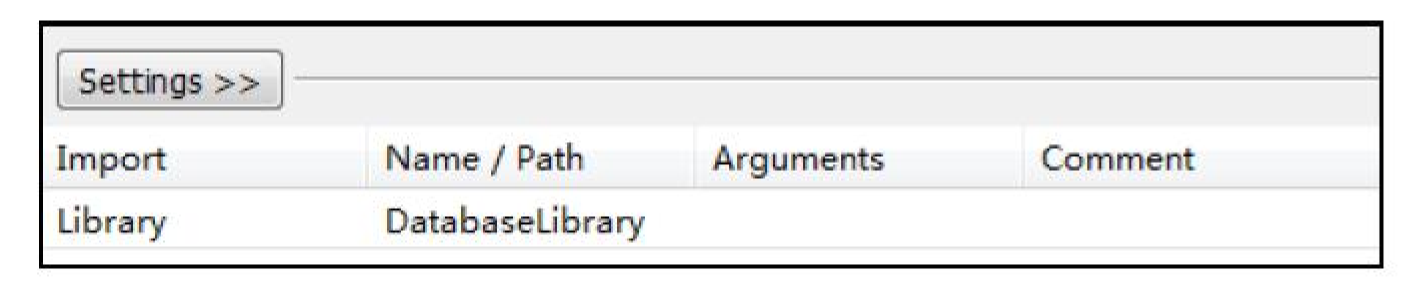


图 6-2-7

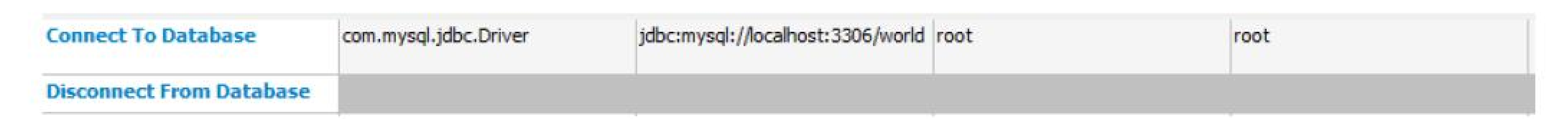
将编译生成好的 robotframework-dblibrary-3.2-SNAPSHOT.jar 包和 MySQL 的 Driver 驱动包一起加入运行环境的 CLASSPATH 下面，然后在测试用例集中引入 DatabaseLibrary 库，如图 6-2-8 所示。


Settings >>

Import	Name / Path	Arguments	Comment
Library	DatabaseLibrary		

图 6-2-8

引入后，我们就可以正常使用 jython 的方式来调用 Java 语言版本的 DatabaseLibrary 库了，如图 6-2-9 所示。



Connect To Database	com.mysql.jdbc.Driver	jdbc:mysql://localhost:3306/world	root	root
Disconnect From Database				

图 6-2-9

使用 Java 语言版本的 DatabaseLibrary 库中的关键字时，和使用 Python 语言版本的 DatabaseLibrary 库中部分关键字的传参有些不一样。Java 语言版本的 DatabaseLibrary 连接数据库时是通过 jdbc 的方式来进行的。这里我们列举一些 Java 语言版本的 DatabaseLibrary 库连接常用的示例，如表 6-2-1 所示。Java 语言版本的 Connect To Database 关键字接收 [数据库 driver 类|jdbc 连接地址|数据库用户名|数据库密码| alias=default]五个参数。alias 如果不传入的话，默认取名为 default。

表 6-2-1　DatabaseLibrary 库链接一些常用数据库的示例

Connect To Database	com.mysql.jdbc.Driver	jdbc:mysql://localhost:3306/world	root	root	连接 MySQL 数据库
Connect To Database	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@servername:port:dbname	system	12345678	连接 Oracle 数据库
Connect To Database	org.h2.Driver	jdbc:h2:mem:robotframeworkt;DB_CLOSE_DELAY=-1	sa	sa	连接 h2 数据库库

这里我们看一个 Java 语言实现的 DatabaseLibrary 库源码的片段，大家可以了解一下 Java 语言版本中是如何实现和封装关键字的。这是 Connect To Database 关键字的实现方法，我们可以看到是通过在方法上加 Java 注解的方式来实现的。@RobotKeyword 和@ArgumentNames 注解均来自 javalib-core 库，我们自己在开发时可以通过 maven 依赖的方式引入。

```
<dependency>
   <groupId>org.robotframework</groupId>
   <artifactId>javalib-core</artifactId>
   <version>1.2.1</version>
</dependency>
```

DatabaseLibrary 库的源码片段：

```
@RobotKeyword("Establish the connection to the database. This is mandatory before
any of"
      + "the other keywords can be used and should be ideally done during the "
      + "suite setup phase. To avoid problems ensure to close the connection again
"
      + "using the disconnect-keyword.\n\n"
      + "It must be ensured that the JAR-file containing the given driver can be
"
      + "found from the CLASSPATH when starting robot. Furthermore it must be "
      + "noted that the connection string is database-specific and must be valid
of course.\n\n"
      + "If alias is given, connection can be later referred with that. If alias
was in use, existing connection "
      + "is replaced with new one\n\n" + "" + "Example: \n"
      + "| Connect To Database | com.mysql.jdbc.Driver |
jdbc:mysql://my.host.name/myinstance | UserName | ThePassword | default |")
@ArgumentNames({ "Driver class name", "Connection string", "Database username",
"Database password","Database alias=default" })
public void connectToDatabase(String driverClassName, String connectString,
String dbUser, String dbPassword,
      String... aliasParam)
      throws SQLException, InstantiationException, IllegalAccessException,
ClassNotFoundException {
   String alias = aliasParam.length > 0 ? aliasParam[0] : defaultAlias;
   Class.forName(driverClassName).newInstance();
   setConnection(DriverManager.getConnection(connectString, dbUser,
dbPassword), alias);
}
```

除了通过 Jython 语言来调用 Java 的 Library 外，我们还可以通过 Remote 的方式来实现调用 Java 语言实现的 Library 库。从 https://github.com/ombre42/jrobotremoteserver 链接中可以获取到用 Java 语言来实现的远程接口服务，如图 6-2-10 所示。

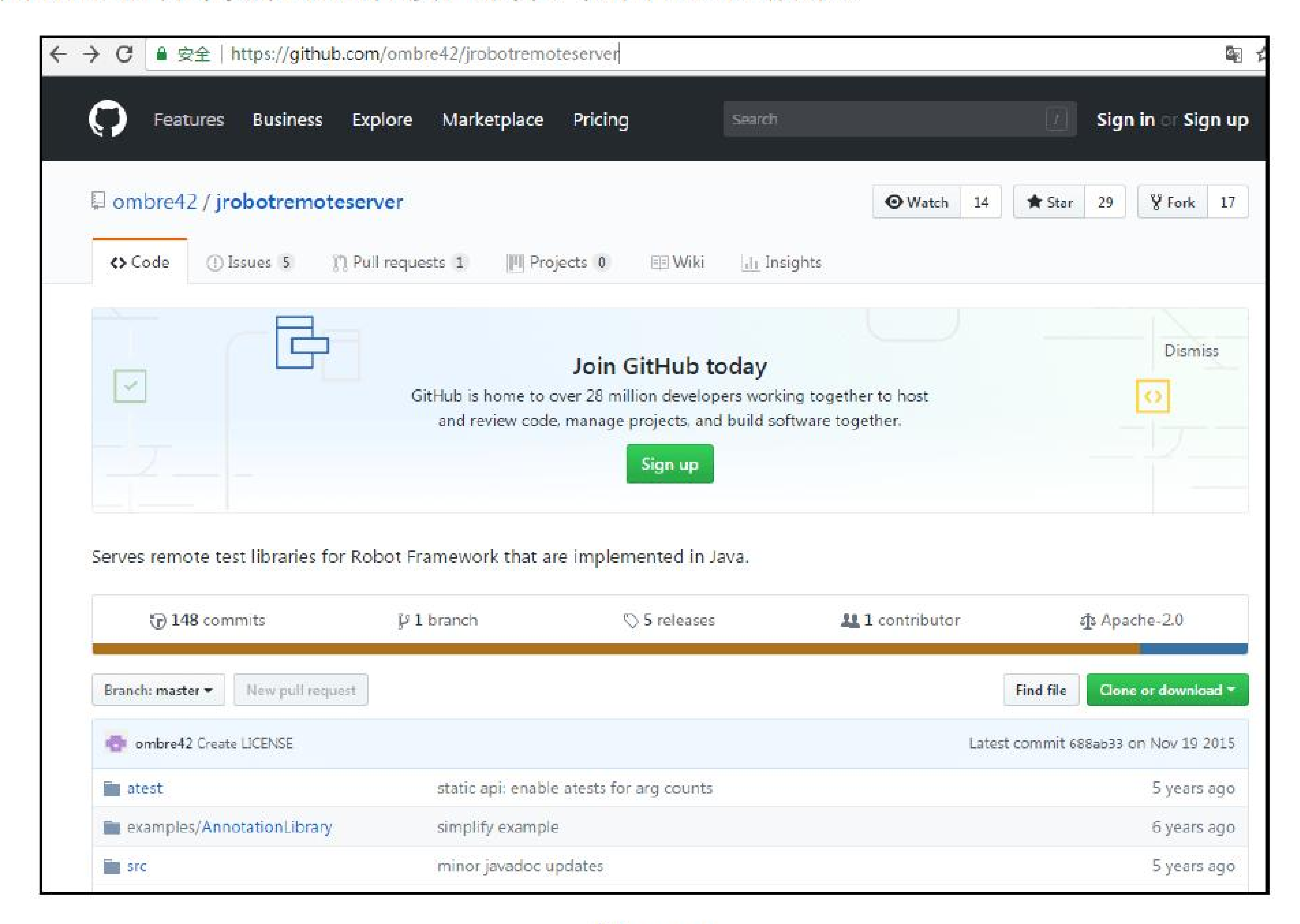


图 6-2-10

我们不建议在实际工作中过多地使用 Jython 的方式来调用 Java 语言实现的 Library，因为每个测试用例在执行时都需要一个启动 jvm 虚拟机的过程，会比使用 Python 语言的方式执行耗时更长，而且兼容性并不是十分好，推荐使用 Remote 的方式来调用 Java 语言实现的 Library 库。

6.2.2 使用 Java 编写自定义的 Lib

前面我们已经讲了如何调用 Java 语言编写的 Lib 库，这里我们将介绍一下如何使用 Java 语言编写自定义的 Lib 库。

要使用 Java 编写自定义 Lib 库，我们首先需要构建一个 Java 的开发工程。这里以构建一个 Java 的 maven 项目为例。在构建的 maven 工程的 pom.xml 文件中，我们需要引入如下必要依赖。

```
<dependency>
   <groupId>org.robotframework</groupId>
   <artifactId>javalib-core</artifactId>
   <version>0.9.1</version>
</dependency>
```

这个是必须引入的依赖包，作用是提供了创建 RobotFramework 关键字的注解等功能，方

便我们快速创建一个 Lib 库以及对应的关键字。

```
<dependency>
   <groupId>com.github.ombre42</groupId>
   <artifactId>jrobotremoteserver</artifactId>
   <version>2.0-BETA</version>
</dependency>
```

这是 Remote Server 的依赖包，作用是可以启动一个 RobotFramework 的远程调用接口服务，然后在 RIDE 中通过 Remote 的方式就可以连接到该远程调用服务上，如果不使用远程调用服务，就可以不引入该依赖包。

```
    <plugin>
      <groupId>com.googlecode.robotframework-maven-plugin</groupId>
      <artifactId>robotframework-maven-plugin</artifactId>
      <version>1.1.1</version>
      <executions>
         <execution>
            <phase>test</phase>
            <goals>
               <goal>run</goal>
            </goals>
         </execution>
      </executions>
      <configuration>
         <variables>
            <variable>BUILDING:True</variable>
         </variables>
      </configuration>
   </plugin>
</plugins>
```

这是 Robot Framework 提供的 maven 插件，不是一个必需的 maven 工程依赖。该插件的作用在于可以模拟 RIDE 来执行测试用例，在使用 maven 编译打包时可以同时执行 Robot Framework 的测试用例。通过如下方式来指定测试用例的位置。

```
<testResources>
   <testResource>
      <directory>src/test/resources</directory>
      <filtering>true</filtering>
   </testResource>
</testResources>
```

在 maven 工程构建好了后，就可以编写自定义的 Lib 库了。表 6-2-2 中描述了使用 Java 语言来编写自定义 Lib 时一些常用的 Java 注解。

表 6-2-2　常用的 Java 注解

注解名称	使用描述
@RobotKeywords	该注解一般用于 Java 类的头部，用来标注该 Java 类提供的是一个 Robot Framework 关键字类
@RobotKeyword	该注解和@RobotKeywords 注解需要一起配合使用。@RobotKeyword 注解一般用于 Java 中某个具体方法的头部，用来标注该方法提供的是一个 Robot Framework 关键字。可以通过@RobotKeyword 后面加括号的方式来说明该关键字的用途，比如@RobotKeyword("这是一个示例关键字")
@ArgumentNames	该注解需要和@RobotKeyword 注解一起使用，用于标注一个 Robot Framework 关键字需要传入的参数。示例：@ArgumentNames({"elementString"})

【示例 1】使用 Java 的方式来实现 Robot Framework 中 Sting Lib 库（如图 6-2-11 所示）的部分关键字 Convert To Lowercase 和 Convert To Uppercase。

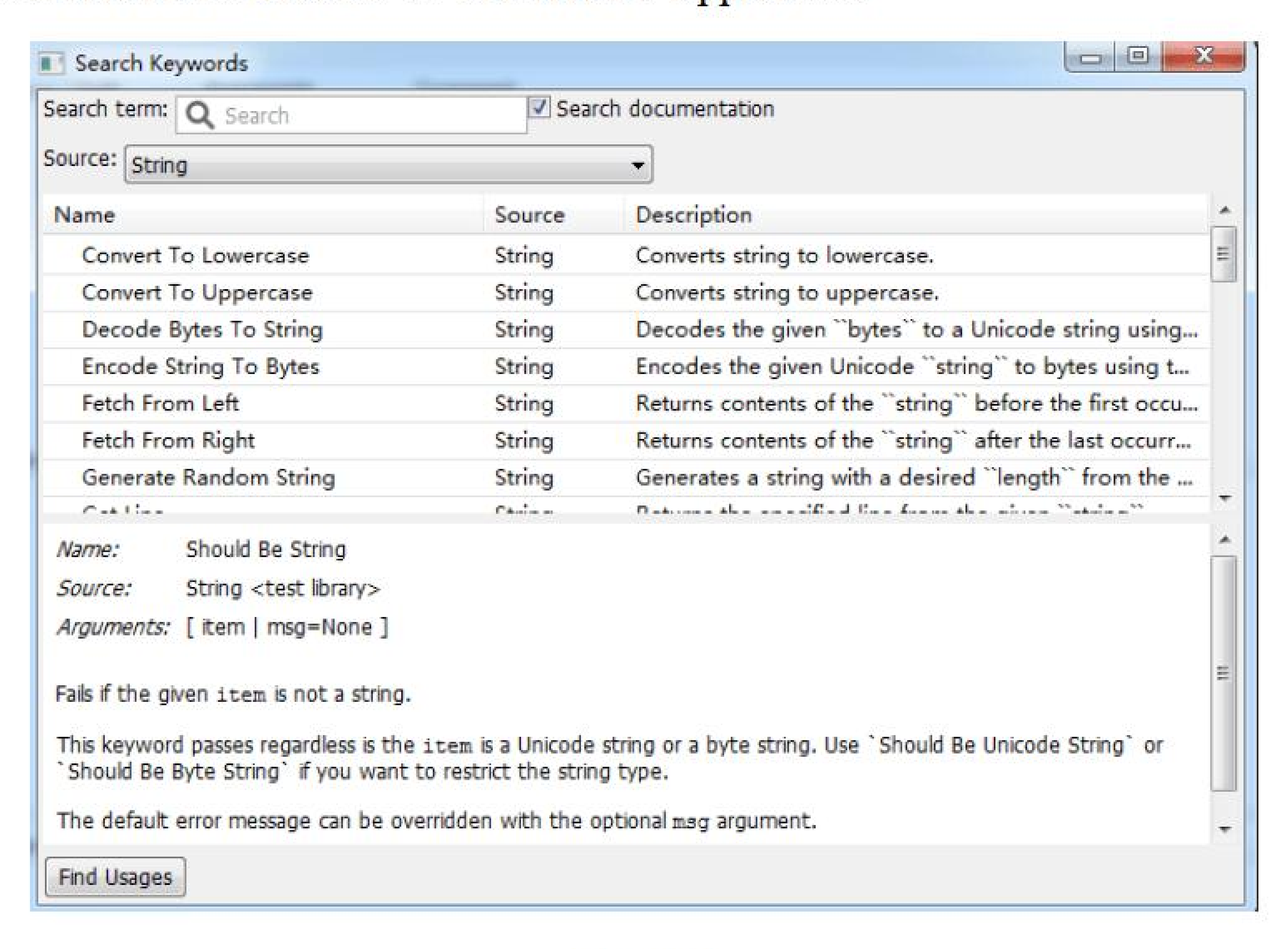


图 6-2-11

```
package com.example.keywords;

import org.robotframework.javalib.annotation.ArgumentNames;
import org.robotframework.javalib.annotation.RobotKeyword;
import org.robotframework.javalib.annotation.RobotKeywords;



@RobotKeywords
public class StringKeyWord {

    @RobotKeyword("Convert To Lowercase")
    @ArgumentNames({"string"})
    public String convertToLowercase(String string) {
```

```
        System.out.print("Convert "+string+" To Lowercase");
        return string.toLowerCase();
    }

    @RobotKeyword("Convert To Uppercase")
    @ArgumentNames({"string"})
    public String convertToUppercase(String string){
        System.out.print("Convert "+string+" To Uppercase");
        return string.toUpperCase();
    }

}
```

关键字编写完了之后，我们还需要定义一个 Library 库（继承 AnnotationLibrary 类），并且通过 RemoteServer 的方式来启动。

```
package com.example;
import java.io.IOException;
import java.io.InputStream;
import java.io.StringWriter;
import java.nio.charset.Charset;

import org.apache.commons.io.IOUtils;
import org.robotframework.javalib.library.AnnotationLibrary;
import org.robotframework.remoteserver.RemoteServer;

public class MyRemoteLibrary extends AnnotationLibrary {
    public MyRemoteLibrary() {
        // 关键字类的路径
        super("com/example/keywords/*.class");
    }

    @Override
    public String getKeywordDocumentation(String keywordName) {
        if (keywordName.equals("__intro__"))
            return getIntro();
        return super.getKeywordDocumentation(keywordName);
    }

    /**
     * 远程接口服务的启动
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        RemoteServer.configureLogging();
        RemoteServer server = new RemoteServer();
        //向 server 中加入自定义的 library，并且设置远程接口服务的端口
        server.addLibrary(MyRemoteLibrary.class, 8270);
        server.start();
    }
```

```
    /**
     * 定义 Library 的说明信息
     * @return
     */
    private String getIntro() {
        try {
            InputStream introStream =
MyRemoteLibrary.class.getResourceAsStream("__intro__.txt");
            StringWriter writer = new StringWriter();
            IOUtils.copy(introStream, writer, Charset.defaultCharset());
            return writer.toString();
        }
        catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

定义完 Library 后，就可以以 RemoteServer 的方式来启动远程接口服务，如图 6-2-12 所示。

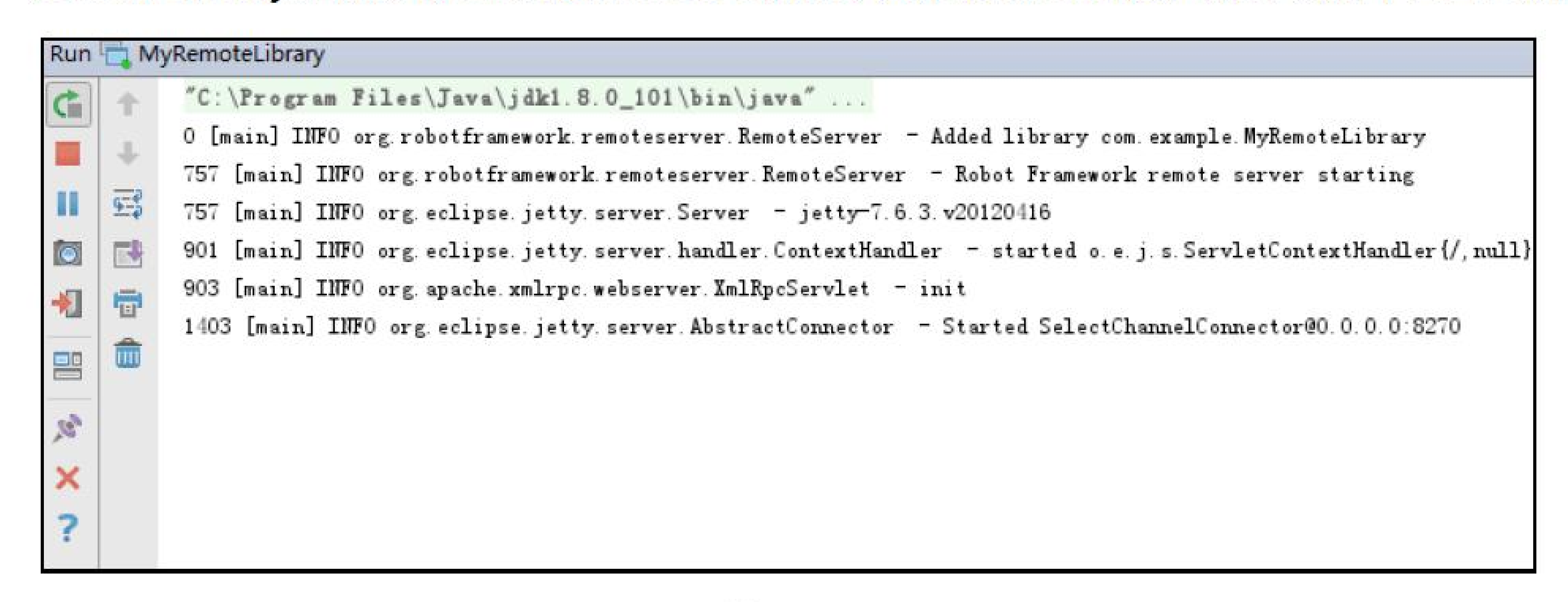


图 6-2-12

在 RIDE 中，我们通过 Remote 连接到启动的远程接口服务中，如图 6-2-13 所示。

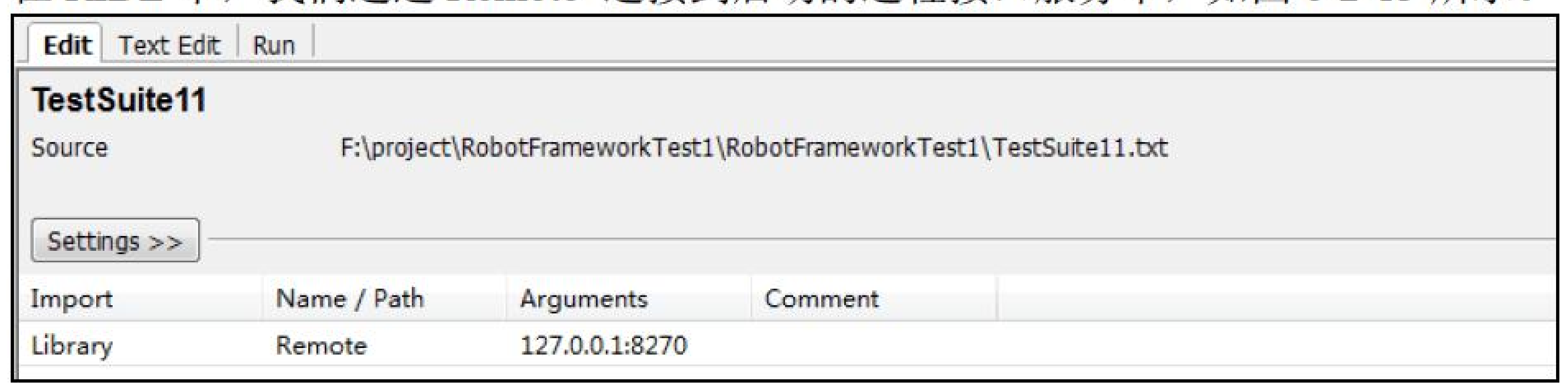


图 6-2-13

然后按 F5 快捷键，就可以看到远程服务中定义的关键字了，如图 6-2-14 所示。

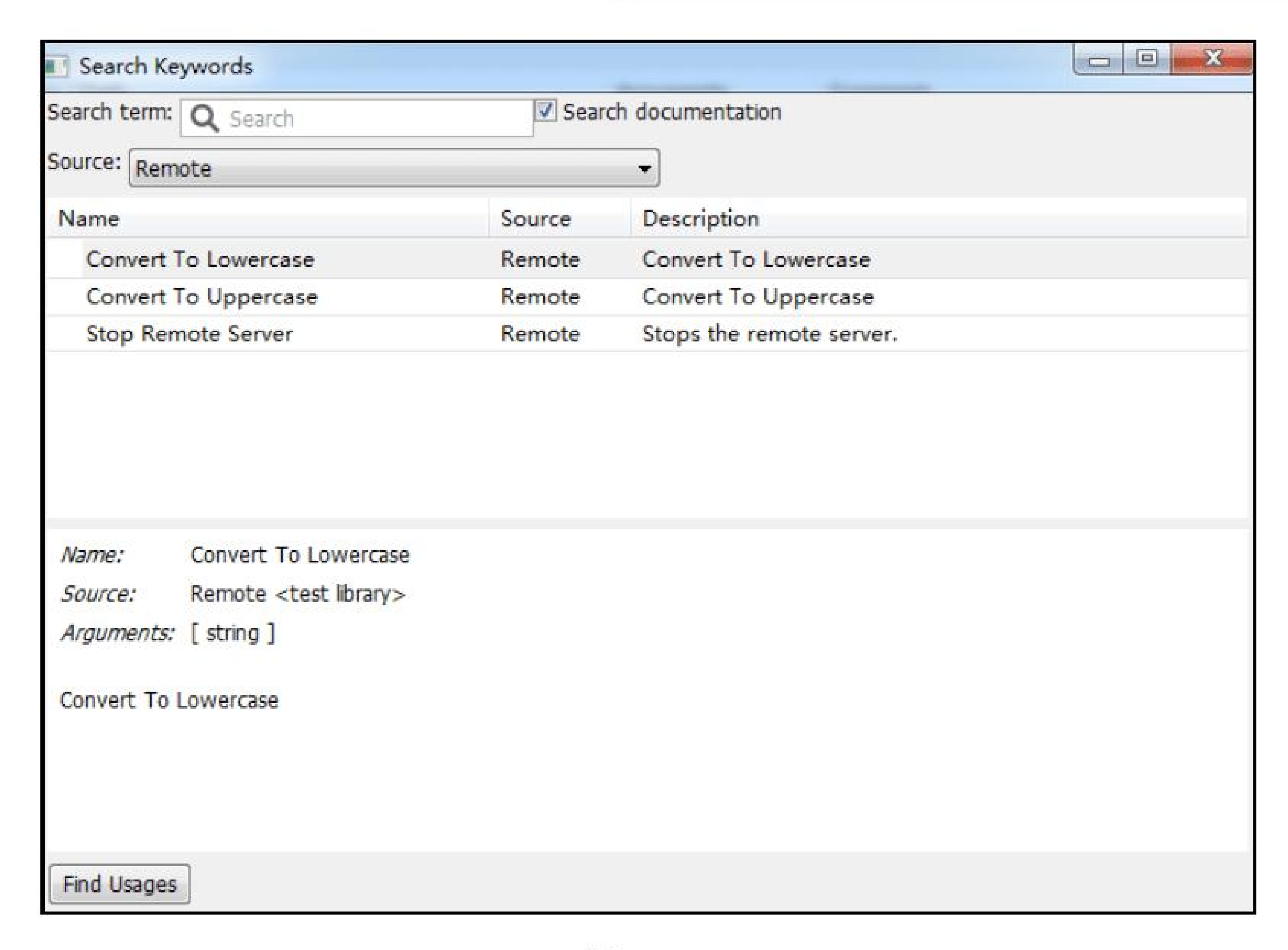


图 6-2-14

【示例 2】调用远程接口服务中定义的关键字，如图 6-2-15 所示。

${strLowercase }	Convert To Lowercase	robotFramework
log	${strLowercase }	
${strUppercase }	Convert To Uppercase	${strLowercase }
log	${strUppercase }	

图 6-2-15

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite11.TestCase001
20180822 10:04:23.328 :  INFO : Convert robotFramework To Lowercase
20180822 10:04:23.328 :  INFO : ${strLowercase } = robotframework
20180822 10:04:23.330 :  INFO : robotframework
20180822 10:04:23.337 :  INFO : Convert robotframework To Uppercase
20180822 10:04:23.338 :  INFO : ${strUppercase } = ROBOTFRAMEWORK
20180822 10:04:23.340 :  INFO : ROBOTFRAMEWORK
Ending test:   RobotFrameworkTest1.TestSuite11.TestCase001
```

从运行结果看，可以成功调用到我们远程接口服务中自定义编写的两个关键字。

上面我们通过 RemoteServer 的方式来调用 Java 编写的自定义的关键字。我们也可以改用 jybot 的方式来调用 Java 编写的自定义关键字。在工程中，我们引入 maven-assembly-plugin 这个插件，通过执行 mvn clean assembly:assembly -Dmaven.test.skip=true 在打包时将所有相关的依赖包打在一个 jar 包中。这样我们在执行时就不需要手动地一个个去配置执行时需要依赖的其他相关 jar 包了。

```
<properties>
<!--  定义打包时的字符集格式 -->
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <testLibraryClass>MyRemoteLibrary</testLibraryClass>
</properties>

<plugin>
   <artifactId>maven-assembly-plugin</artifactId>
   <configuration>
      <descriptorRefs>
         <descriptorRef>jar-with-dependencies</descriptorRef>
      </descriptorRefs>
      <archive>
         <manifest>
   <!--  配置执行时的 java main 方法 -->
            <mainClass>${testLibraryClass}</mainClass>
         </manifest>
      </archive>
   </configuration>
   <executions>
      <execution>
         <id>make-my-jar-with-dependencies</id>
         <phase>package</phase>
         <goals>
            <goal>single</goal>
         </goals>
      </execution>
   </executions>
</plugin>
```

另外需要注意是，我们需要将上面定义的 MyRemoteLibrary 类移动到 maven 工程的根目录下，如图 6-2-16 所示。

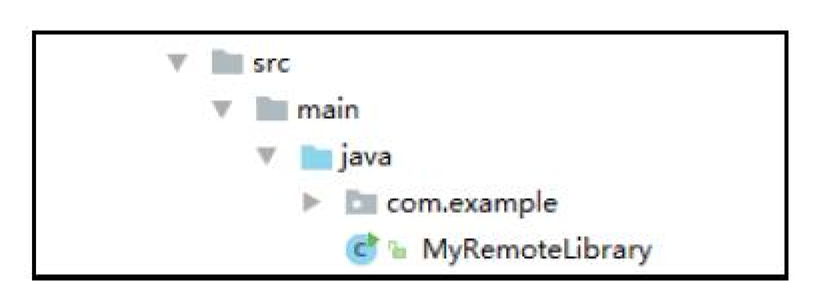


图 6-2-16

执行 mvn clean assembly:assembly -Dmaven.test.skip=true 打包后，就可以生成我们需要的 jar 包了，如图 6-2-17 所示。



名称	修改日期	类型	大小
archive-tmp	2018/8/22 10:22	文件夹	
classes	2018/8/22 10:22	文件夹	
maven-archiver	2018/8/22 10:22	文件夹	
maven-status	2018/8/22 10:22	文件夹	
MyRemoteLibrary-1.0.jar	2018/8/22 10:22	Executable Jar File	7 KB
MyRemoteLibrary-1.0-jar-with-dependencies.jar	2018/8/22 10:22	Executable Jar File	4,692 KB

图 6-2-17

将 MyRemoteLibrary-1.0-jar-with-dependencies.jar 放置到 Java 的 classpath 目录下，我们就

可以在 RIDE 中引入 MyRemoteLibrary 库了，如图 6-2-18 所示。

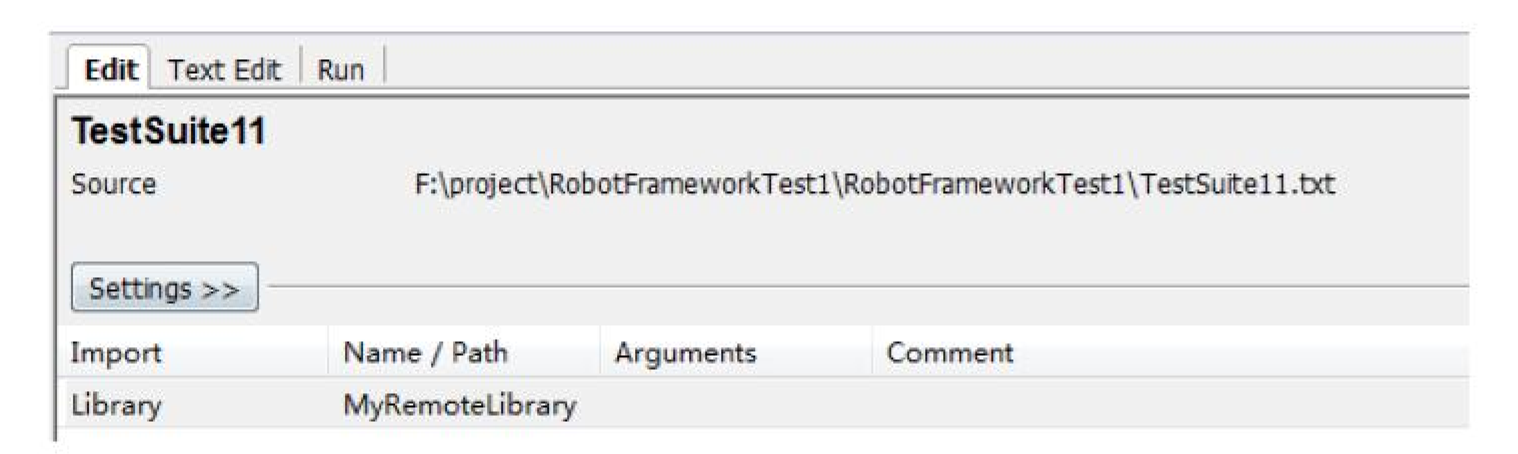


图 6-2-18

引入后，我们再用 jybot 的方式执行上面 RemoteServer 运行时的同样示例，如图 6-2-19 所示。

${strLowercase }	**Convert To Lowercase**	robotFramework
log	${strLowercase }	
${strUppercase }	**Convert To Uppercase**	${strLowercase }
log	${strUppercase }	

图 6-2-19

运行结果如图 6-2-20 所示。

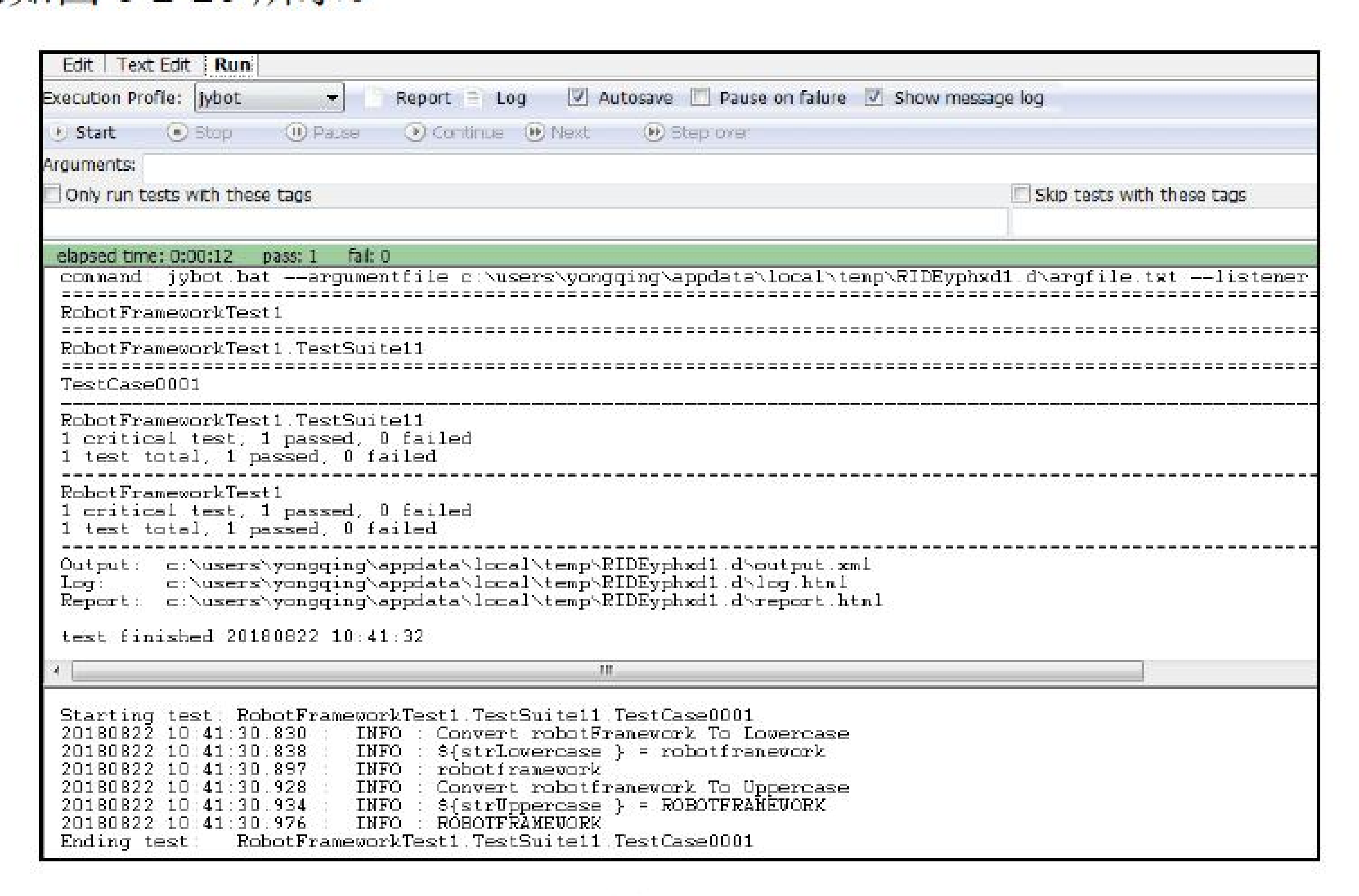


图 6-2-20

可以看到，运行后得到了同样的结果。

第 7 章

自动化测试用例的编写技巧

7.1 自动化测试用例的常用技巧

7.1.1 自动化测试用例的容错

自动化测试用例在实际执行时都可能出现执行失败，但是有时候就是因为某一条测试用例执行失败了，从而影响到整个测试用例集都会执行失败或者浏览器资源无法得到释放。此时测试用例的容错就会变得非常重要。

我们先来看一下 Selenium2Library 自动化测试库中 Register Keyword To Run On Failure 关键字这个例子。

```
Open Browser    https://www.baidu.com/  chrome
Register Keyword To Run On Failure Close Browser
Mouse down  id=kw111
Close Browser
```

在这里，我们故意设置了一个肯定会执行失败的场景，因为在执行 Mouse down 这个关键字时，通过 locator 为 id=kw111 在页面中根本定位不到任何元素，如图 7-1-1 所示。此时问题就来了，测试用例肯定会执行失败，然后案例就会退出，但是退出时并不会将浏览器关闭、释放资源。

此时 Register Keyword To Run On Failure 关键字的设置就非常关键了，它可以在执行失败时让 Selenium2 做关闭浏览器操作。

在实际执行上面的那条测试用例步骤时，我们可以看到使用了这个执行容错关键字后浏览器是可以关闭的。如果没有这个关键字，那么在执行失败时浏览器根本不会关闭。

```
elapsed time: 0:00:16    pass: 0    fail: 1
==============================================================================
RobotFrameworkTest1
==============================================================================
RobotFrameworkTest1.TestSuite6
==============================================================================
TestCase0030                                                          | FAIL |
ERROR: Element id=kw111 not found.
------------------------------------------------------------------------------
RobotFrameworkTest1.TestSuite6                                        | FAIL |
1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed
==============================================================================
RobotFrameworkTest1                                                   | FAIL |
1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed
==============================================================================
Output:  c:\users\yongqing\appdata\local\temp\RIDEvyfrbm.d\output.xml
Log:     c:\users\yongqing\appdata\local\temp\RIDEvyfrbm.d\log.html
Report:  c:\users\yongqing\appdata\local\temp\RIDEvyfrbm.d\report.html

test finished 20180729 15:15:08

Starting test: RobotFrameworkTest1.TestSuite6.TestCase0030
20180729 15:14:53.899 :  INFO : Opening browser 'chrome' to base url 'https://www.baidu.com/'
20180729 15:15:06.635 :  INFO : Close Browser will be run on failure.
20180729 15:15:06.636 :  INFO : Simulating Mouse Down on element 'id=kw111'
20180729 15:15:07.859 :  FAIL : ERROR: Element id=kw111 not found.
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0030
```

图 7-1-1

表 7-1-1 中列出了 Robot Framework 中常用的和容错相关的关键字。

表 7-1-1　Robot Framework 中常用的和容错相关的关键字

关键字	使用描述
Register Keyword To Run On Failure	在 Selenium2Library 库和 AppiumLibrary 库中都有一个这样的关键字，可以帮助我们在使用 Selenium2 或者 Appium 执行对浏览器或者移动手机操作时在执行失败的情况下可以做的一些容错处理
Run Keyword And Continue On Failure	这个关键字来源于 BuiltIn 库，这个库是默认自动加载的，不需要通过 import 的方式来导入。这个关键字可以指定操作执行失败时继续往下执行
Run Keyword If Any Critical Tests Failed 和 Run Keyword If Any Tests Failed 以及 Run Keyword If Test Failed	这几个关键字同样来源于 BuiltIn 库。这个库默认可以自动加载，不需要通过 import 关键字来导入指定库。这些关键字主要用于在关键的测试执行失败时指定需要执行的关键字操作，并且该关键字只能使用在测试用例集 Suite 中。在 Robot Framwork 的每一个测试用例集 suite 中都可以进行 Suite Setup（对测试用例集的初始执行操作，只会执行一遍）和 Suite Teardown（Run Keyword If Any Critical Tests Failed 关键字一般就可以用于此处）设置 在每一个 Suite 中，我们还可以设置 Test SetUp（该测试用例集下的每一个案例在初始执行时都会执行的初始化操作）和 Test Teardown（该测试用例集下的每一个测试用例在执行结束时都会执行的操作）以及 Test TimeOut（该测试用例集下的每一个自动化测试用例执行超时时可以设置执行的操作）处理
Run Keyword If Timeout Occurred	该关键字和上面的那几个类似。这个关键字主要是解决案例执行超时，可以指定在超时发生时执行何种关键字操作，在处理容错时经常会用到。这个关键字用于在 Robot Framwork 每一个测试用例集 Suite 的 Suite Teardown 设置中

7.1.2 自动化测试用例的测试数据初始化和脏数据的处理

在自动测试中，数据的初始化操作以及案例执行完成后需要对部分脏数据进行清理，以免影响下一次测试用例的执行操作或者对别的测试用例的执行造成影响。

（1）通过执行数据库脚本来进行数据库中数据的初始化操作和脏数据的清理。

在 Robot Framework 中，DatabaseLibrary 库中的 Execute Sql Script 关键字和 Execute Sql String 关键字都可以用来完成数据初始化操作和脏数据的清理操作。

（2）通过 OperatingSystem 库来对自动化测试用例执行过程需要的文件数据做初始化操作，以及执行完成后文件脏数据的清理。

表 7-1-2 中列出了 RobotFramework 中常用的和数据清理相关的关键字。

表 7-1-2 RobotFramework 中常用的和数据清理相关的关键字

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Append To Environment Variable</td><td>该关键字用来增加一个操作系统的环境变量，接收[name | *values | **config]这些参数。示例：<table><tr><td>Append To Environment Variable</td><td>NAME</td><td>Robot</td></tr><tr><td>Should Be Equal</td><td>%{NAME}</td><td>Robot</td></tr></table></td></tr>
<tr><td>Remove Environment Variable</td><td>该关键字用于删除指定的系统环境变量，接收[*names]多个参数，即可以支持删除多个环境变量。示例：<table><tr><td>Remove Environment Variable</td><td>env1</td><td>env2</td></tr></table></td></tr>
<tr><td>Copy Directory</td><td>该关键字用于做文件夹的复制操作，接收[source | destination]两个参数，如果 destination 参数对应的文件夹已经存在，就将 source 参数对应的文件夹复制到 destination 参数对应的文件夹下面，否则复制过去后直接创建一个新文件夹。示例：<table><tr><td>Copy Directory</td><td>/home/robottest1</td><td>/home/robottest2</td></tr></table></td></tr>
<tr><td>Copy File</td><td>该关键字用于复制文件操作，接收[source | destination]两个参数，如果 destination 参数对应的文件已经存在，就直接覆盖，否则直接复制过去创建一个新文件。示例：<table><tr><td>Copy File</td><td>/home/robottest1.txt</td><td>/home/robottest2.txt</td></tr></table></td></tr>
<tr><td>Copy Files</td><td>该关键字用于复制多个文件操作，接收[*sources_and_destination]多个参数。复制时，可以传入多个文件参数，但是最后一个参数必须是一个目标文件夹路径。示例：<table><tr><td>Copy Files</td><td>/home/robottest1.txt</td><td>/home/robottest2.txt</td><td>/home/robottest3.txt</td><td>/opt/</td></tr></table></td></tr>
<tr><td>Move Directory</td><td>该关键字用于做文件夹的移动操作，接收[source | destination]两个参数。示例：<table><tr><td>Move Directory</td><td>/home/test1</td><td>/home/test2</td></tr></table></td></tr>
</table>

（续表）

<table>
<tr><th>关键字</th><th>使用描述</th></tr>
<tr><td>Move File</td><td>该关键字用于做文件的移动操作，接收[source | destination]两个参数，示例：<table><tr><td>Move File</td><td>/home/robottest1.txt</td><td>/opt/</td></tr></table></td></tr>
<tr><td>Move Files</td><td>该关键字用于做多个文件的移动操作，接收[*sources_and_destination]多个参数。可以移动多个文件，但是传入的最后一个参数必须是一个目标文件夹路径。示例：<table><tr><td>Move Files</td><td>/home/robottest1.txt</td><td>/home/robottest2.txt</td><td>/opt/</td></tr></table></td></tr>
<tr><td>Append To File</td><td>该关键字用于向指定文件内追加内容，接收[path | content | encoding=UTF-8]三个参数，如果指定的路径文件不存在，就新创建一个文件，如果文件已经存在，就直接追加内容，并且可以指定追加内容的字符集格式，默认为 UTF-8。示例：<table><tr><td>Append To File</td><td>/opt/test.txt</td><td>Robot framework</td></tr></table></td></tr>
<tr><td>Remove Directory</td><td>该关键字用来删除一个指定的文件夹目录，接收[path | recursive=False]两个参数。recursive 参数用来判断是否需要进行递归删除，设置为否时，当指定路径下存在子路径或者文件时将无法删除。示例：<table><tr><td>Remove Directory</td><td>/home/test</td></tr></table></td></tr>
<tr><td>Remove File</td><td>该关键字用来删除指定的文件，接收[path]一个参数。示例：<table><tr><td>Remove File</td><td>/opt/test.txt</td></tr></table></td></tr>
<tr><td>Remove Files</td><td>该关键字用来删除多个指定的文件，接收[*paths]多个参数。示例：<table><tr><td>Remove Files</td><td>/home/robottest1.txt</td><td>/home/robottest2.txt</td></tr></table></td></tr>
</table>

OperatingSystem 库还有 Count Directories In Directory、Count Files In Directory、Count Items In Directory、Create Binary File、Create Directory、Create File、Empty Directory、Get File、Get File Size、Get Modified Time、Grep File、Join Path、Join Paths、List Directories In Directory、List Directory、List Files In Directory、Touch、Split Path、Split Extension、Directory Should Be Empty、Directory Should Exist、Directory Should Not Be Empty、Directory Should Not Exist、File Should Be Empty、File Should Exist、File Should Not Be Empty、File Should Not Exist、Split Extension 等关键字，在我们做文件类数据的初始化和数据清理时能提供很大的帮助。

7.2 如何高效地维护好自动化测试用例

7.2.1 提取出共用变量统一维护

在自动化测试中，经常需要使用到变量，也经常需要定义变量。在 Robot Framework 中，可以定义单个变量，也可以通过列表或者字典的方式来定义我们需要的变量。

在一个测试用例集中，多个案例都需要用到的变量可以放到测试用例集中进行统一定义和维护。图 7-2-1 所示就是一个测试用例集的界面。通过 Add Scalar、Add List、Add Dict 定义出来的变量都是对整个测试用例集共用的变量。

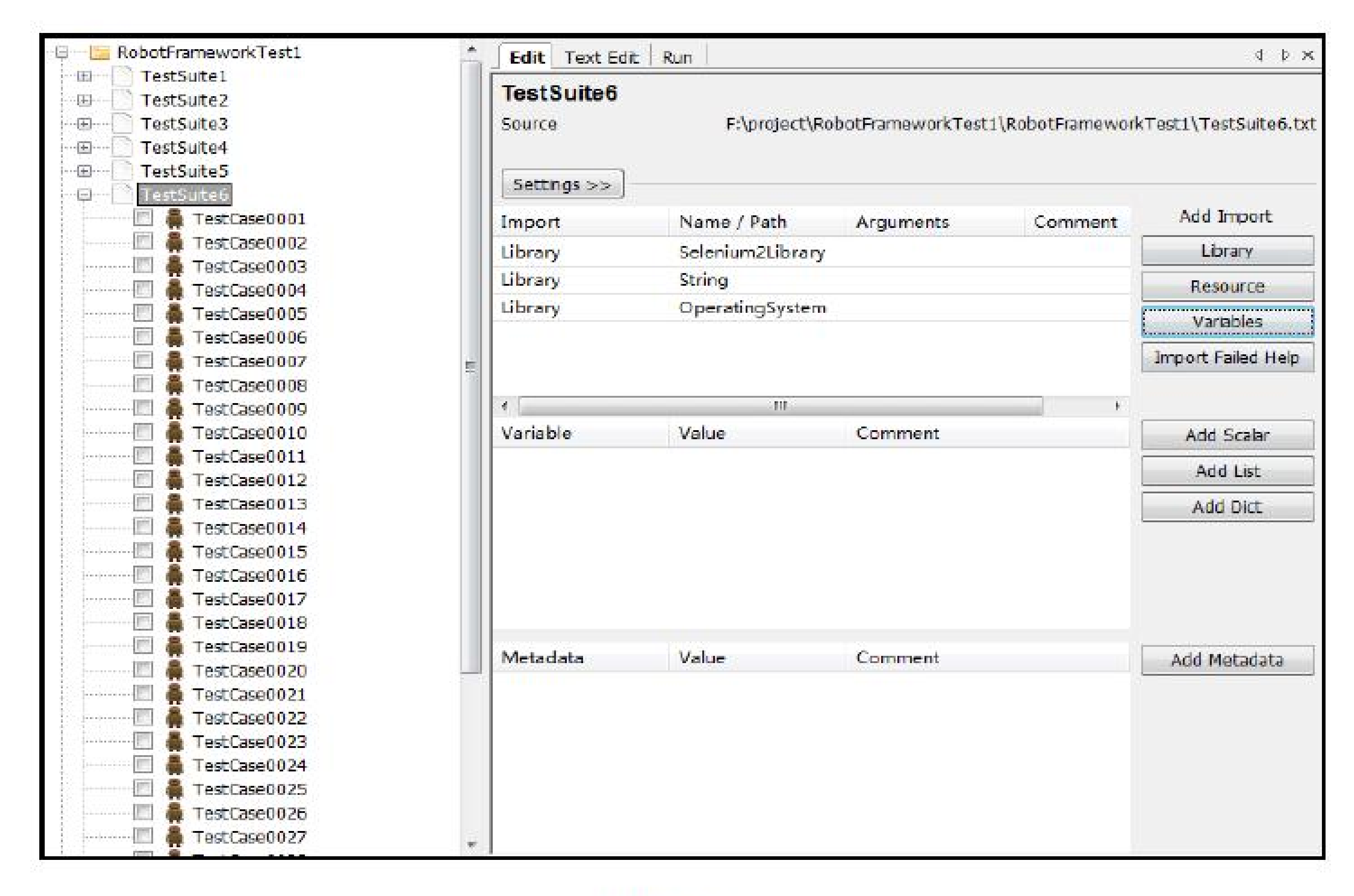


图 7-2-1

【示例 1】用 Add Scala 定义出来的测试用例集都能共用的变量。

定义一个${bookname}变量，如图 7-2-2 所示。

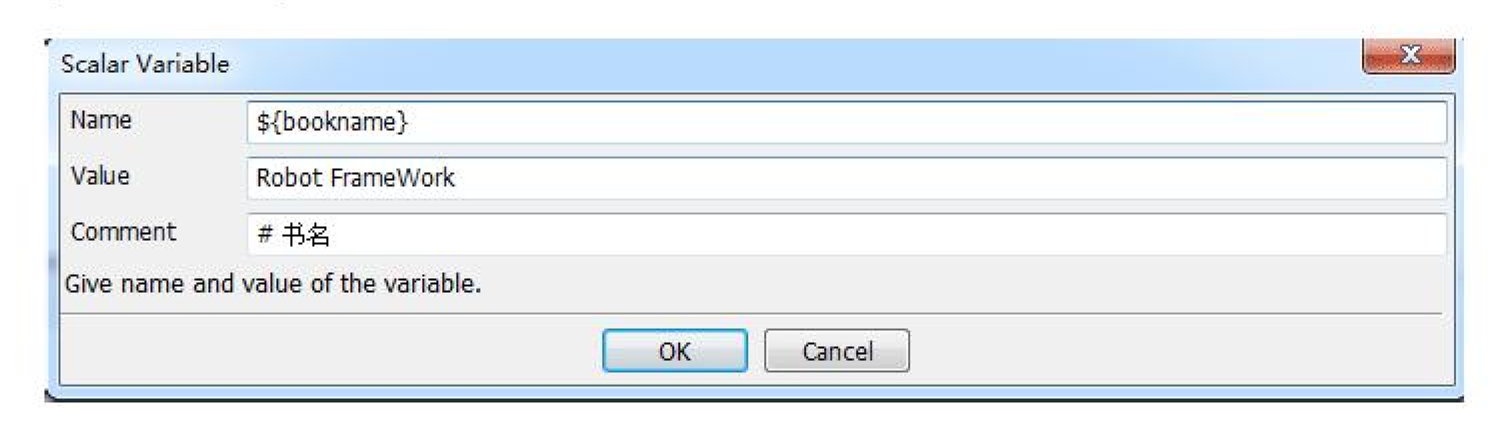


图 7-2-2

定义成功后，界面上会展示定义好的变量，如图 7-2-3 所示。

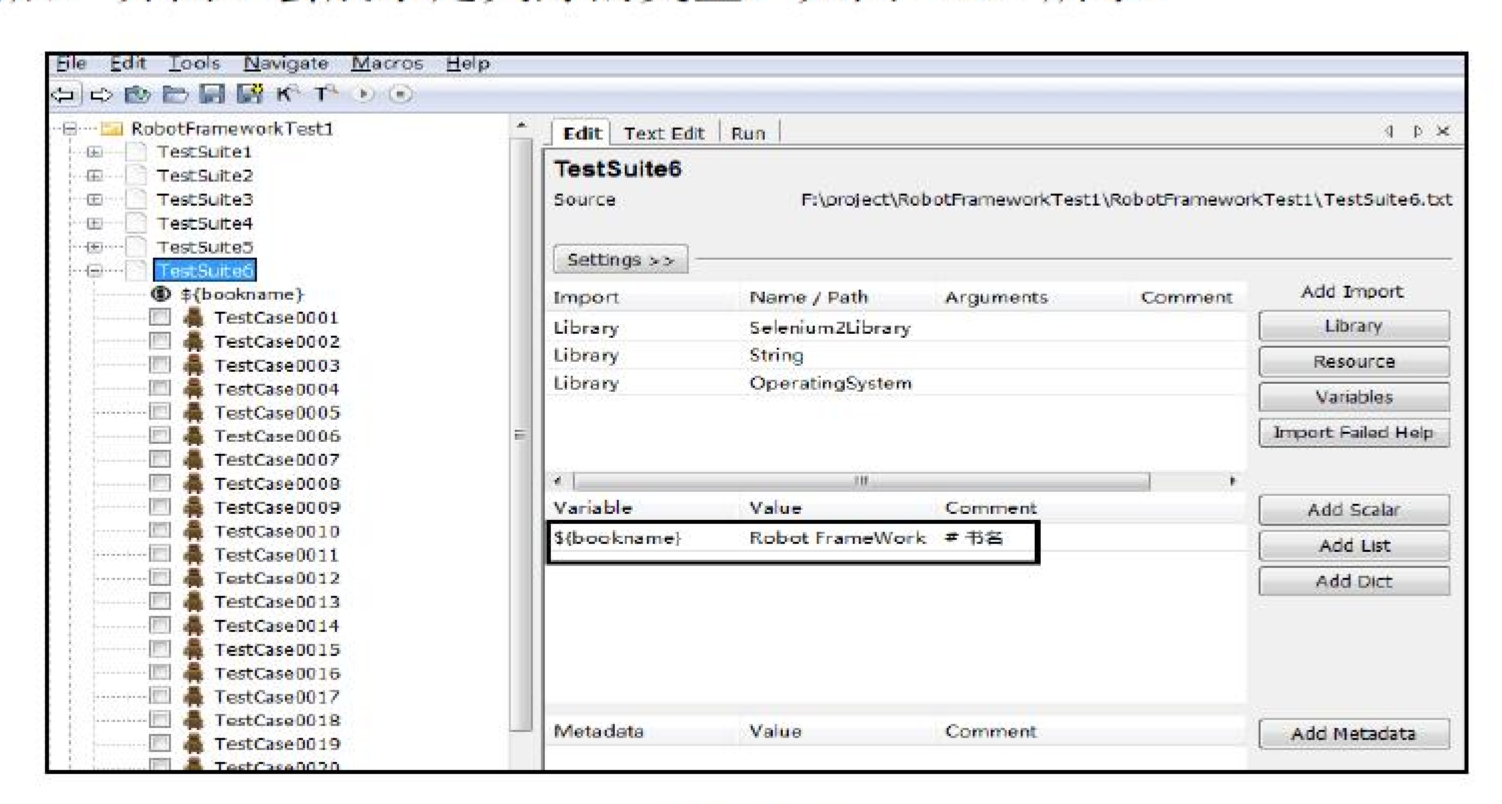


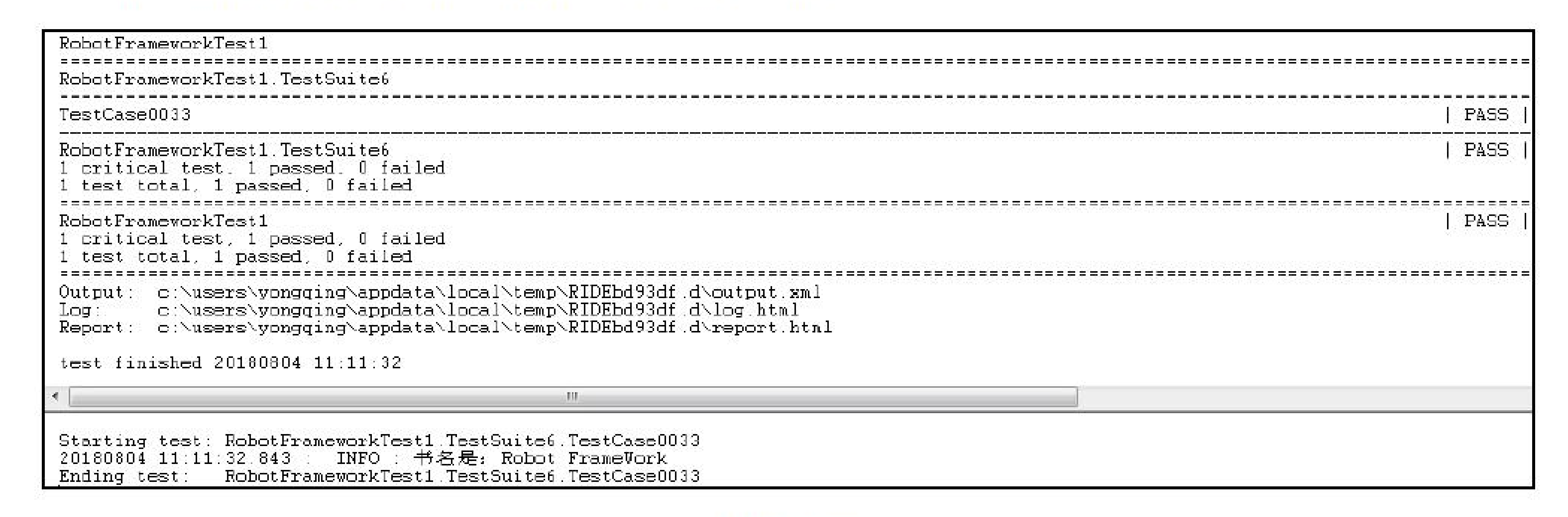
图 7-2-3

定义完成后，再在这个测试用例集下建立一个测试用例来使用变量，如图 7-2-4 所示。

log	书名是：${bookname}	
Should Be Equal	${bookname}	Robot FrameWork

图 7-2-4

运行上面的测试用例后，结果如图 7-2-5 所示。



```
RobotFrameworkTest1
==============================================================================
RobotFrameworkTest1.TestSuite6
------------------------------------------------------------------------------
TestCase0033                                                          | PASS |
------------------------------------------------------------------------------
RobotFrameworkTest1.TestSuite6                                        | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
RobotFrameworkTest1                                                   | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  c:\users\yongqing\appdata\local\temp\RIDEbd93df.d\output.xml
Log:     c:\users\yongqing\appdata\local\temp\RIDEbd93df.d\log.html
Report:  c:\users\yongqing\appdata\local\temp\RIDEbd93df.d\report.html

test finished 20180804 11:11:32

Starting test: RobotFrameworkTest1.TestSuite6.TestCase0033
20180804 11:11:32.843 :  INFO : 书名是：Robot FrameWork
Ending test:   RobotFrameworkTest1.TestSuite6.TestCase0033
```

图 7-2-5

从运行结果可以看到，在测试用例集中定义的共用变量可以在单个测试用例中直接引用。

如果在测试用例中也定义了一个和测试用例集重名的变量，那么这个变量会直接覆盖测试用例集中的已有变量，如图 7-2-6 所示。

${bookname}	Set Variable	Robot FrameWork自动化测试框架核心指南
log	书名是：${bookname}	
Should Be Equal	${bookname}	Robot FrameWork

图 7-2-6

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0034
20180804 11:19:23.920 : INFO : ${bookname} = Robot FrameWork 自动化测试框架核心指南
20180804 11:19:23.921 : INFO : 书名是：Robot FrameWork 自动化测试框架核心指南
20180804 11:19:23.922 : FAIL : Robot FrameWork 自动化测试框架核心指南 != Robot FrameWork
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0034
```

从运行结果看，在测试用例中定义的重名变量已经被直接覆盖掉了。

【示例 2】通过 Add List 来定义测试用例集都共用的 List 变量。

定义一个性别类型的@{sex}List 变量，如图 7-2-7 所示。

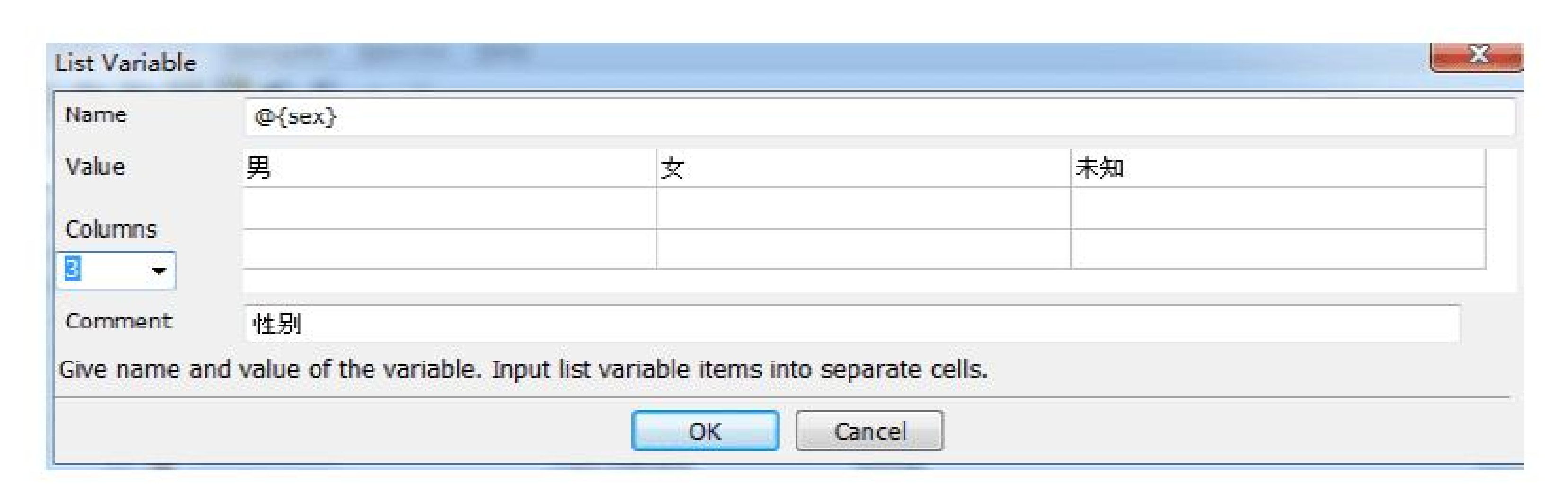


图 7-2-7

定义成功后，界面上可以展示定义好的 List 变量，如图 7-2-8 所示。

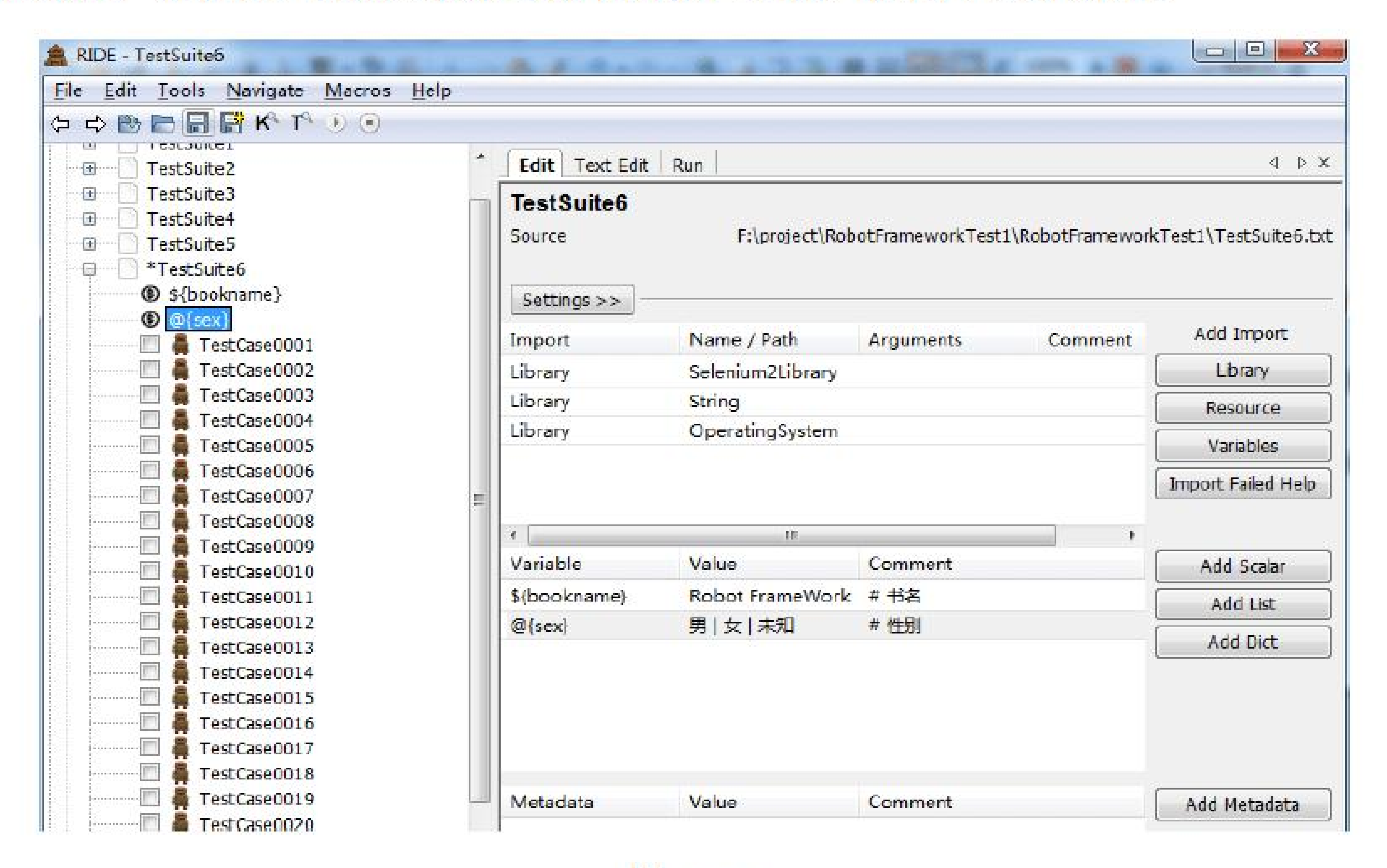


图 7-2-8

我们再写一个简单的测试用例，用来调用测试用例集中定义的共用 List 变量，如图 7-2-9 所示。



图 7-2-9

运行结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite6.TestCase0035
20180804 15:24:07.569 :  INFO : 性别：男
20180804 15:24:07.571 :  INFO : 性别：女
20180804 15:24:07.573 :  INFO : 性别：未知
Ending test:  RobotFrameworkTest1.TestSuite6.TestCase0035
```

7.2.2　在单个自动化测试用例中多使用变量

上面提到了要提取共用的变量统一维护，除共用的变量外，我们在每一个测试用例中也需要尽量多定义自己的变量，而且这些变量尽量定义到案例的开始步骤中，这样后面在维护案例

时就会高效很多。比如一个案例运行的期望值可能会发生变化，那么我们修改的时候就直接修改定义的变量值，而不需要在一大堆很长的步骤中去找。我们来看一个示例，如图 7-2-10 所示。

${expectedTitle}	Set Variable	百度一下	
Open Browser	http://www.baidu.com	chrome	
${title}	Get Title		
Close Browser			
Should Be Equal	${title}	${expectedTitle}	获取到的值：${title} 和期望的值：${expectedTitle}不

图 7-2-10

在这个示例中，由于我们提取了变量，因此后期在维护案例时会方便很多。

7.2.3　提取复用的业务或者步骤，封装自定义的用户关键字

在自动化测试用例中，经常会出现一些步骤是共用的（很多案例中都需要使用的步骤），我们可以把这些步骤提取出来，而不是在每个测试用例中都重复编写这些测试步骤。抽取成共用后，可以封装成自己定义的用户关键字，在其他的测试用例中直接使用这个自定义的用户关键字即可。这样以后共用的步骤发生变化后，就不需要每一个测试用例都去做修改了，只需要修改自定义的用户关键字即可。如图 7-2-11 所示，在 RIDE 界面上选中一个测试用例集后，右击鼠标键，选择“New User Keyword”选项，即可新建一个自定义的用户关键字。

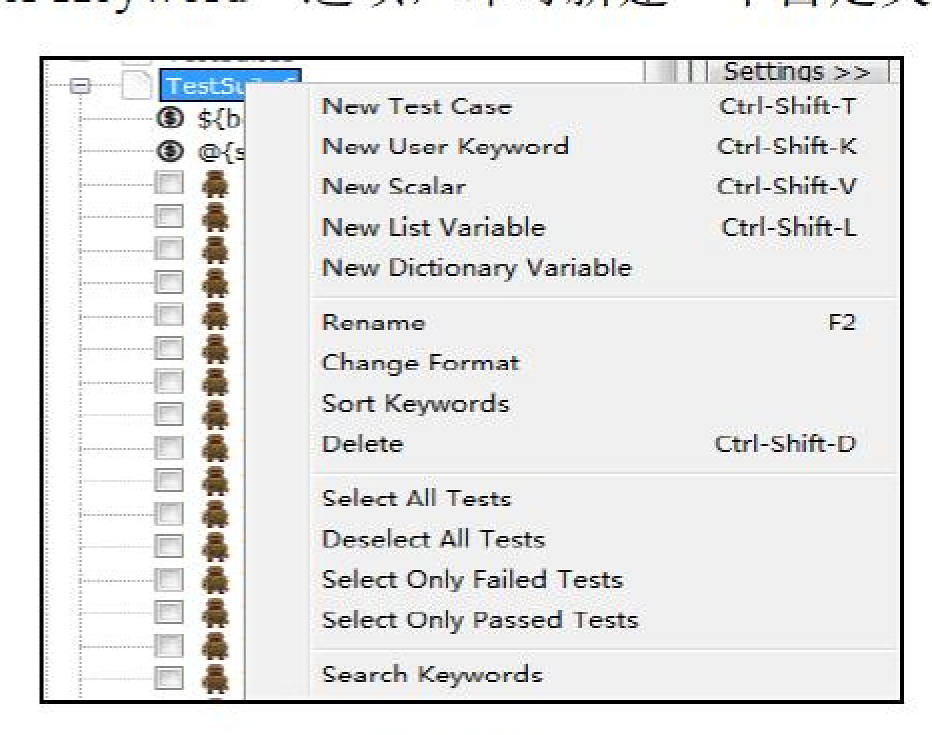


图 7-2-11

下面我们将一个登录百度的操作封装成一个通用的自定义用户关键字。登录本身是一个通用的步骤，很多案例都需要先进行登录，这里将关键字的名字定义为 login baidu。我们让该关键字接收两个参数，用户名${user}和密码${passwd}，因为登录时肯定是需要输入用户名和密码的，如图 7-2-12 所示。

New User Keyword

Name: login baidu

Arguments: ${user}|${passwd}

Give a name and arguments for the new user keyword.
Specify the arguments separated with a pipe character like '${arg1} | ${arg2}'.
Default values are given using equal sign and the last argument can be a list variable.
Example: '${arg1} | ${arg2}=default value | @{rest}'.
Note. You can use variable shortcuts in this field.

OK　Cancel

图 7-2-12

定义成功后，界面展示如图 7-2-13 所示。

login baidu

Settings <<

Documentation: 这是一个登录百度的操作。

Tags: <Add New>

Arguments: ${user} | ${passwd}

Teardown

Return Value

Timeout

图 7-2-13

之后，我们就可以在该关键字中将登录操作步骤加进去，如图 7-2-14 所示。

```
    Open Browser    http://www.baidu.com    chrome
    Click Element    //*[@id="u1"]/a[7]
    Sleep    2
    Click Element    id=TANGRAM_PSP_10_footerULoginBtn
    Input Text    id=TANGRAM_PSP_10_userName    ${user}
    Input Password    id=TANGRAM_PSP_10_password    ${passwd}
Click Button    id=TANGRAM_PSP_10_submit
```

login baidu

Settings <<

Documentation: 这是一个登录百度的操作。

Tags: <Add New>

Arguments: ${user} | ${passwd}

Teardown

Return Value

Timeout

1	Open Browser	http://www.baidu.com	chrome
2	Click Element	//*[@id="u1"]/a[7]	
3	Sleep	2	
4	Click Element	id=TANGRAM_PSP_10_footerUL	
5	Input Text	id=TANGRAM_PSP_10_userNam	${user}
6	Input Password	id=TANGRAM_PSP_10_passwor	${passwd}
7	Click Button	id=TANGRAM_PSP_10_submit	

图 7-2-14

关键字定义好了以后，就可以在测试用例中使用自己定义的关键字了。如图 7-2-15 所示，

我们定义了一个测试用例，并使用 login baidu 自定义的关键字。

login baidu	yongqing_zh	123456
Sleep	10	
Close Browser		

图 7-2-15

执行测试用例，运行结果如图 7-2-16 所示，可以看到调用自定义封装的关键字成功了。

```
Starting test: RobotFrameworkTest1.TestSuite9.TestCase0001
20180804 16:41:20.508 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20180804 16:41:28.557 :  INFO : Clicking element '//*[@id="u1"]/a[7]'.
20180804 16:41:30.712 :  INFO : Slept 2 seconds
20180804 16:41:30.713 :  INFO : Clicking element
'id=TANGRAM__PSP_10__footerULoginBtn'.
20180804 16:41:32.908 :  INFO : Typing text 'yongqing_zh' into text field
'id=TANGRAM__PSP_10__userName'
20180804 16:41:33.105 :  INFO : Typing password into text field
'id=TANGRAM__PSP_10__password'
20180804 16:41:33.282 :  INFO : Clicking button 'id=TANGRAM__PSP_10__submit'.
20180804 16:41:44.326 :  INFO : Slept 10 seconds
Ending test:  RobotFrameworkTest1.TestSuite9.TestCase0001
```

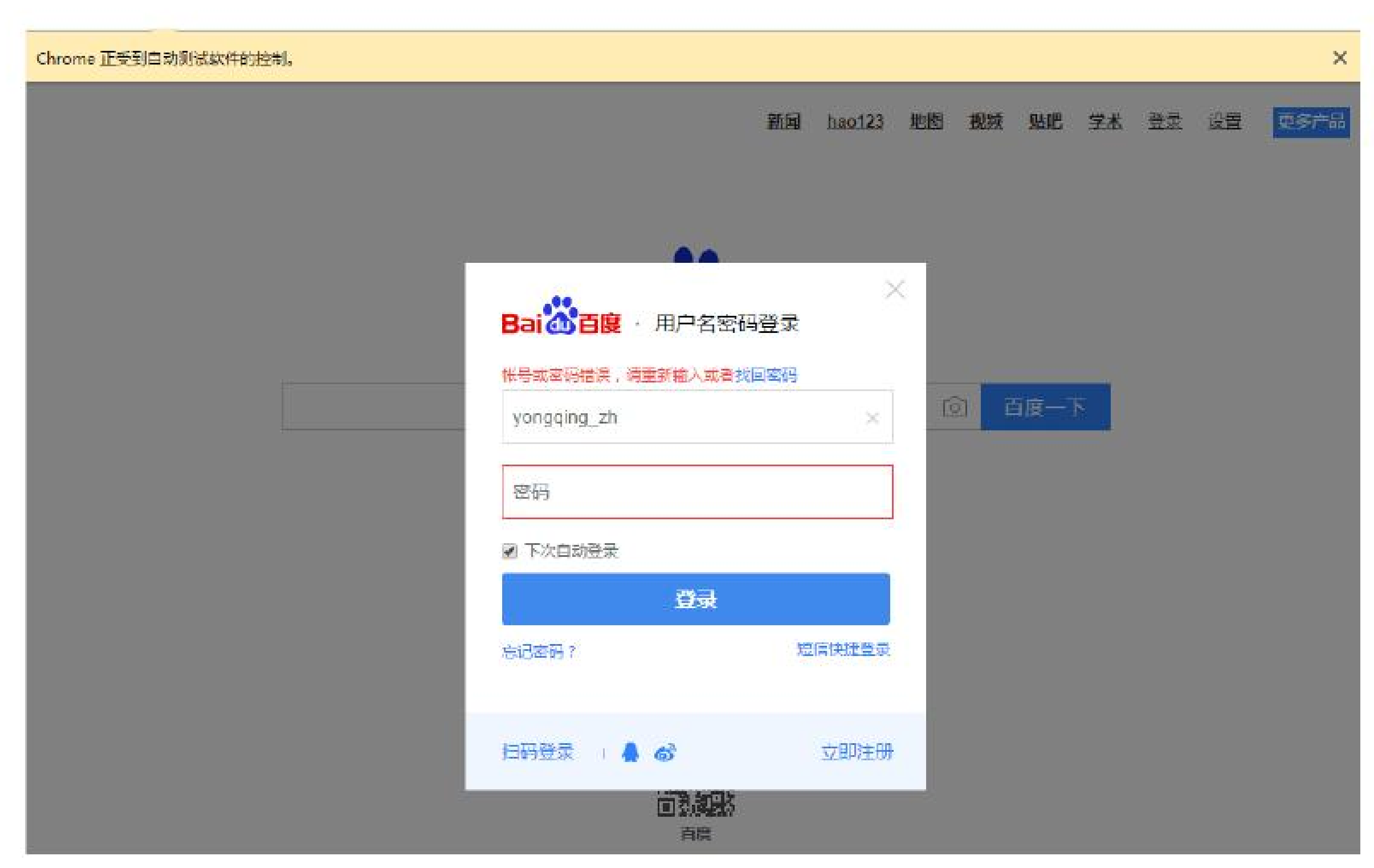


图 7-2-16

7.2.4　封装全局可用的资源文件

从前面可以知道，很多共用的封装都是针对单个测试用例集共用的，那么如何封装多个测试用例集都可以重用的步骤或者变量定义呢？这里就需要用到 Resource 了。选中我们的自动化测试工程，右击鼠标键，选择 New Resource 选项就可以建立一个 Resource 了，如图 7-2-17 所示。

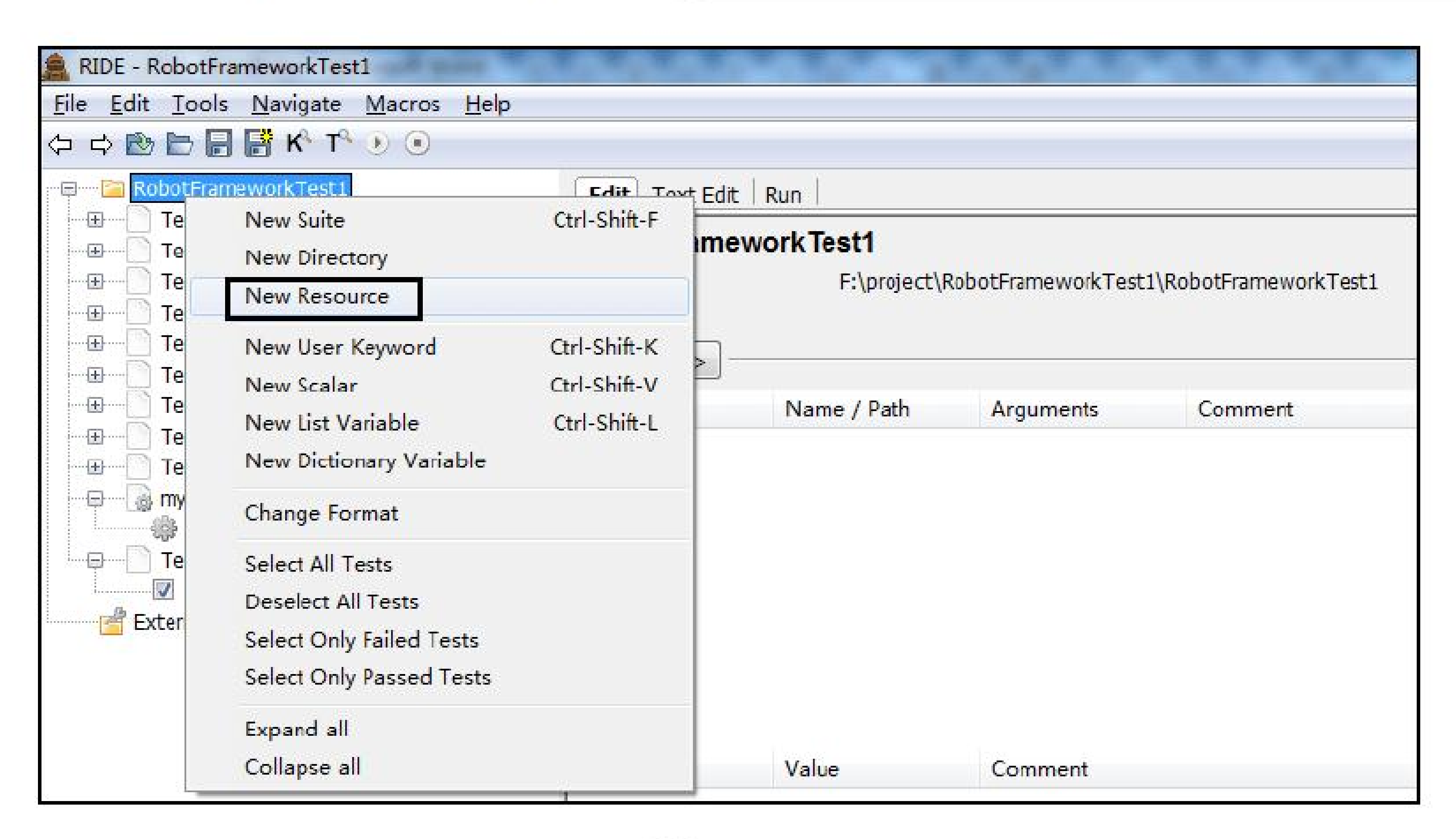


图 7-2-17

在弹出的窗口中输入 Resource 名称就可以定义一个 Resource 了，如图 7-2-18 所示。

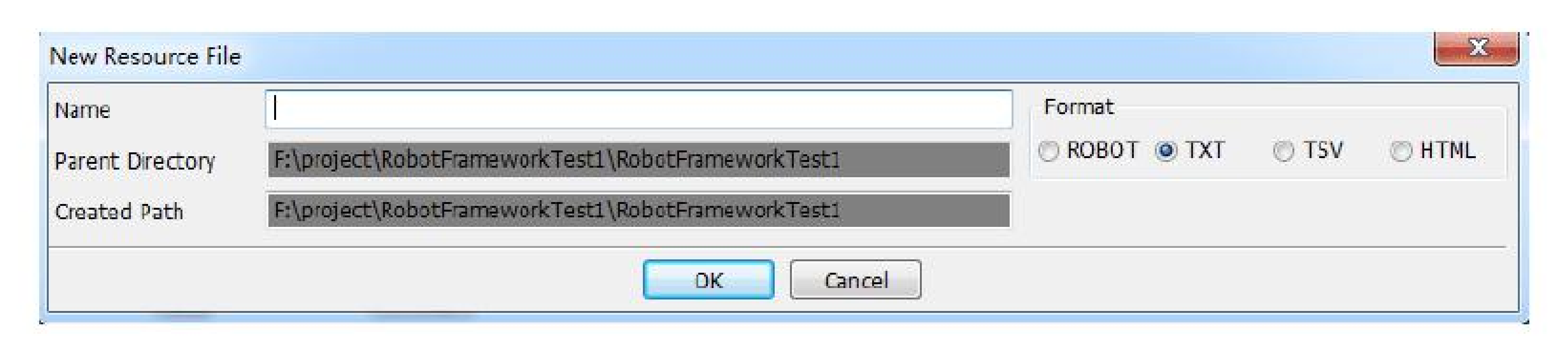


图 7-2-18

定义好一个 Resource 后，界面展示如图 7-2-19 所示。

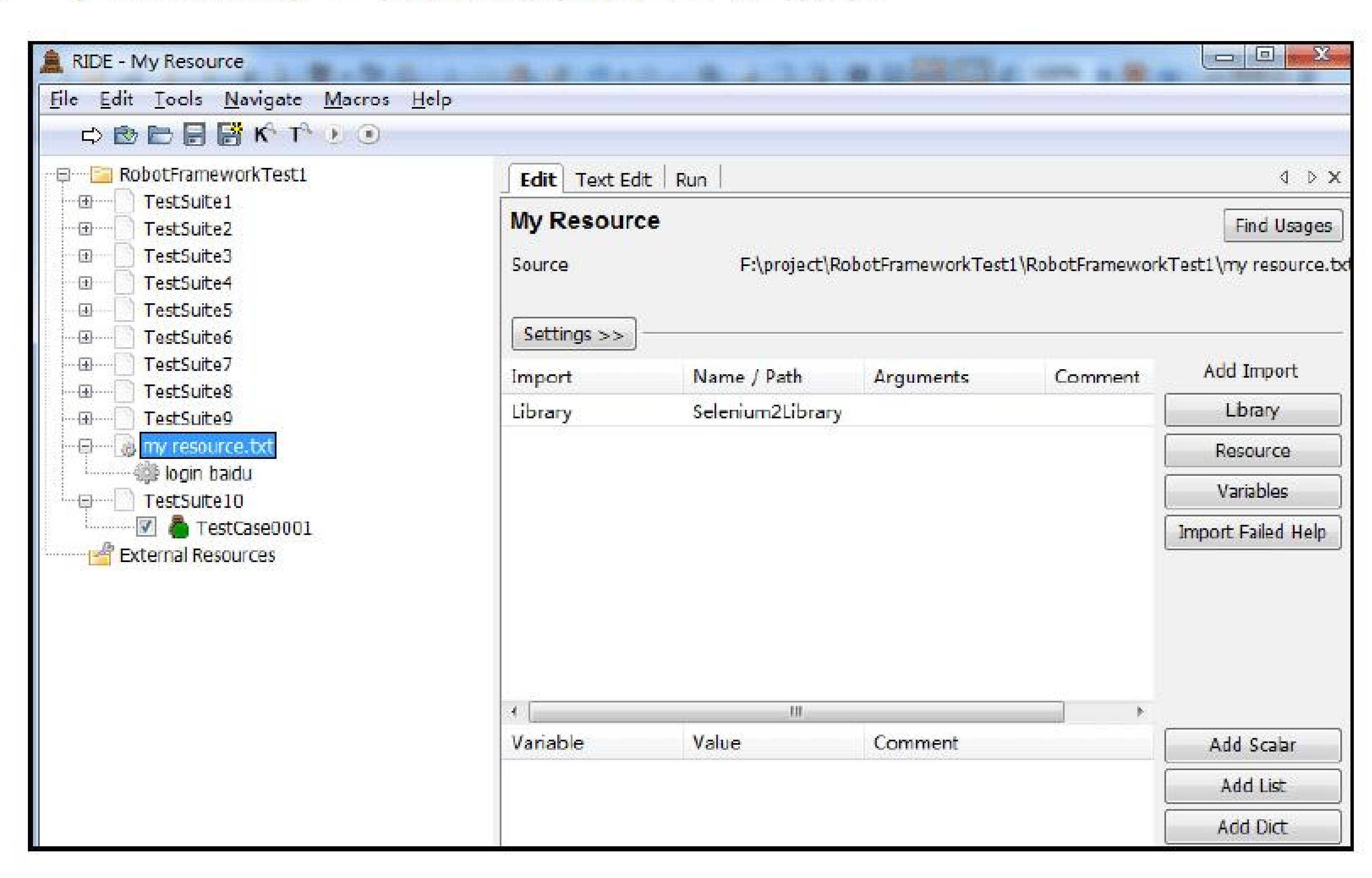


图 7-2-19

在 Resource 中我们可以定义的操作如下：

- 导入需要的 Library 库。在图 7-2-19 中，我们导入了 Selenium2Library 库。
- 引用已经定义好的其他 Resource。

- 定义 Resource 下共用的单个变量、List 变量和 Dictionary 变量。
- 定义 Resource 下自定义的用户关键字。

这里我们将之前定义的 login baidu 关键字复制到我们自己定义的 Resource 下面，如图 7-2-20 所示。

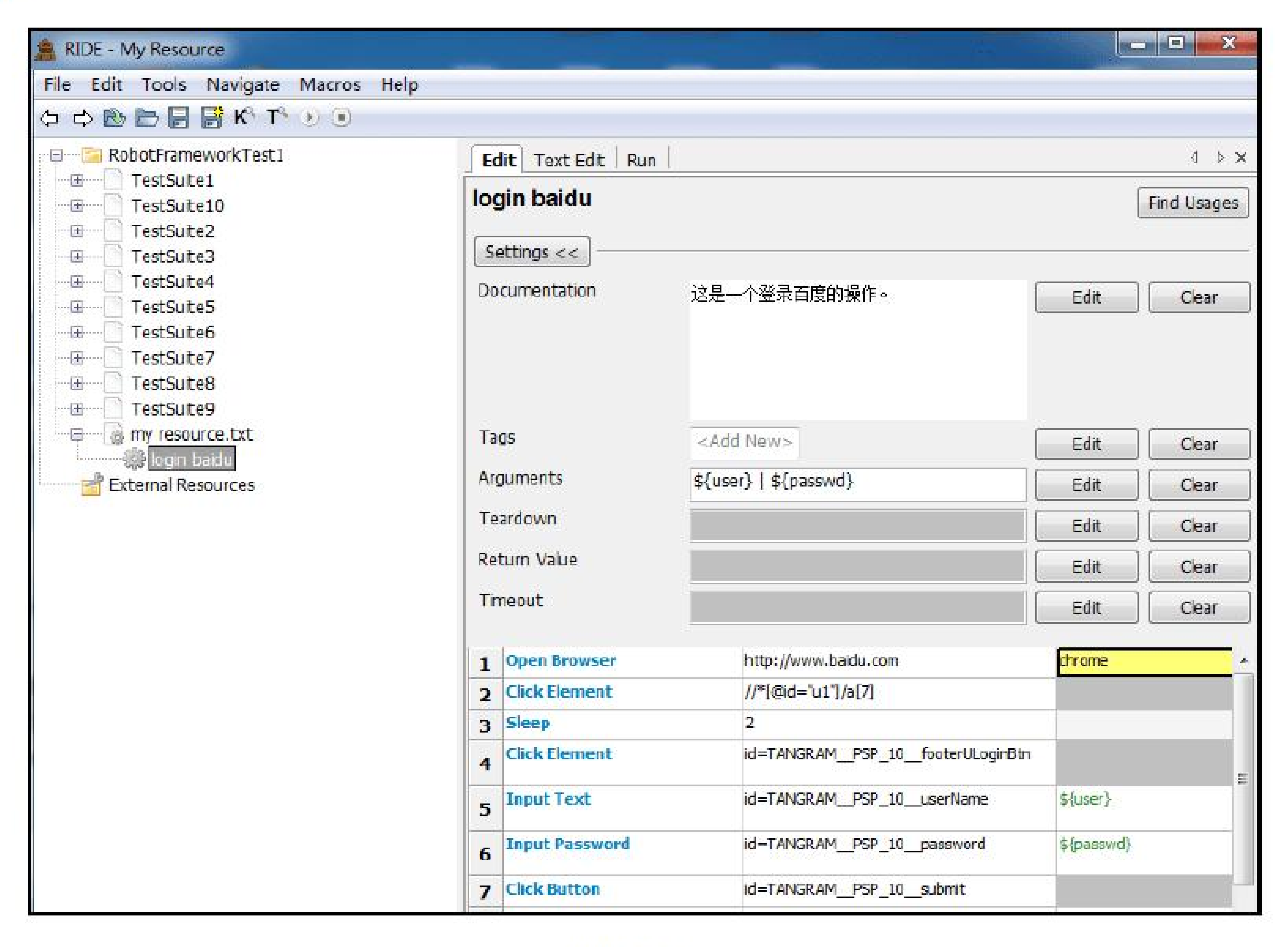


图 7-2-20

然后我们新建一个测试用例集，并且在该测试用例集中引入该 Resource，如图 7-2-21 所示。

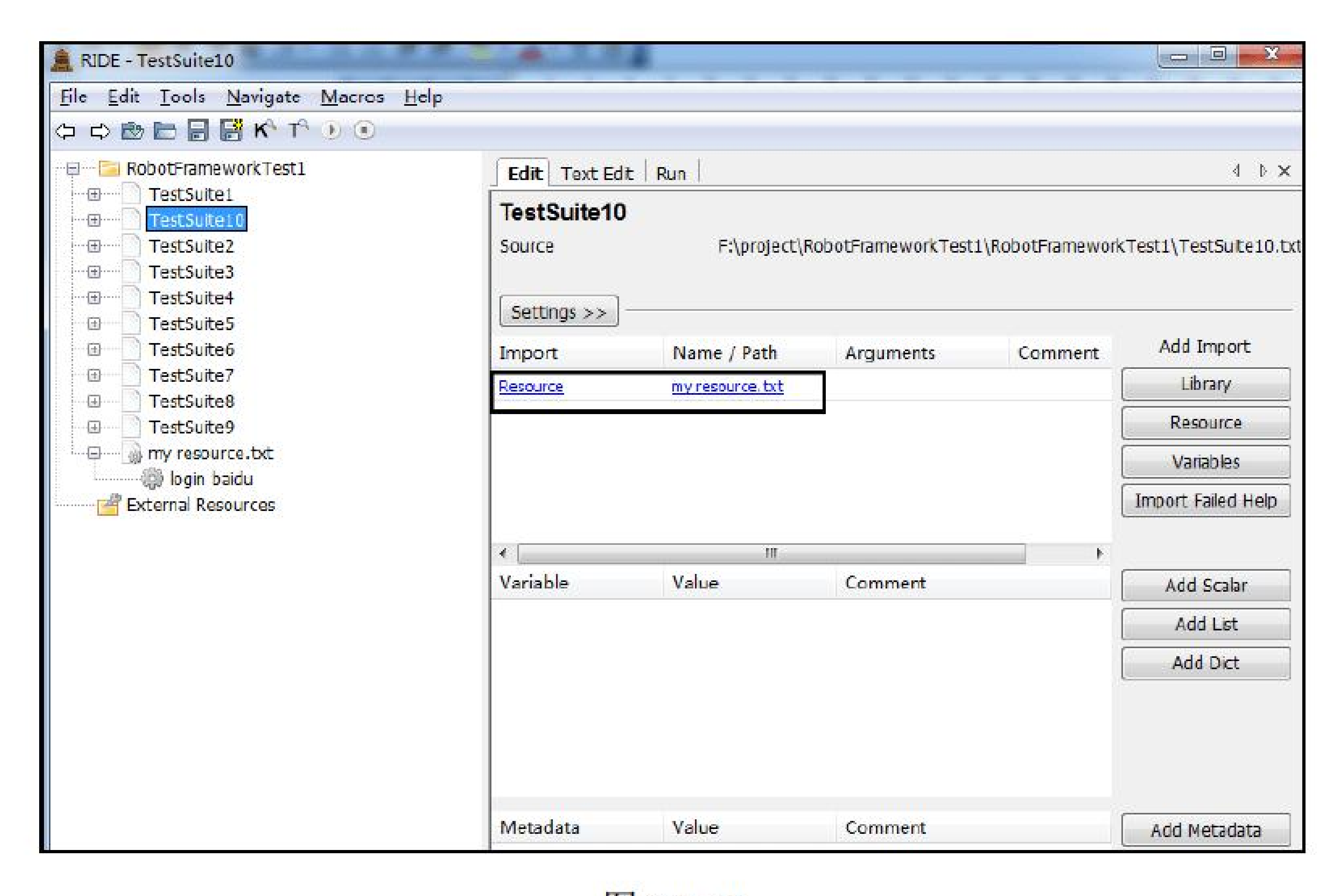


图 7-2-21

在导入 Resource 后，我们就可以使用导入的 Resource 中的关键字了。如图 7-2-22 所示，我们新建了一个测试用例，直接使用 Resource 中的自定义关键字。



图 7-2-22

运行测试用例，结果如下：

```
Starting test: RobotFrameworkTest1.TestSuite10.TestCase0001
20180804 17:18:34.520 :  INFO : Opening browser 'chrome' to base url
'http://www.baidu.com'
20180804 17:18:43.266 :  INFO : Clicking element '//*[@id="u1"]/a[7]'.
20180804 17:18:45.455 :  INFO : Slept 2 seconds
20180804 17:18:45.456 :  INFO : Clicking element
'id=TANGRAM__PSP_10__footerULoginBtn'.
20180804 17:18:47.227 :  INFO : Typing text 'yongqing_zh' into text field
'id=TANGRAM__PSP_10__userName'
20180804 17:18:47.436 :  INFO : Typing password into text field
'id=TANGRAM__PSP_10__password'
20180804 17:18:47.625 :  INFO : Clicking button 'id=TANGRAM__PSP_10__submit'.
20180804 17:18:58.296 :  INFO : Slept 10 seconds
Ending test:  RobotFrameworkTest1.TestSuite10.TestCase0001
```

第 8 章

自动化测试框架的设计

8.1 Jenkins 下自动化测试的调度管理

8.1.1 Jenkins 介绍

Jenkins 是一个功能非常强大的持续集成和持续交付的开源项目，几乎可以处理任何类型的自动构建或者持续集成。Jenkins 可以用于自动化部署，也可以用于自动化测试的调度。通过访问网址 https://jenkins.io/ 可以进入 Jenkins 的官网，如图 8-1-1 所示。



图 8-1-1

Jenkins 是支持常见的 Master-Slave 架构的持续集成项目，很多开源项目都使用了这一常

见的运行架构，比如大数据中常用的 Hadoop 项目等都用了这一运行架构。在 Jenkins 中，Master 节点可以用来负责 Slave 节点的管理、用户提交的 Job 的配置以及把 Job 分发到不同 Slave 节点上进行运行的调度和管理，如图 8-1-2 所示。

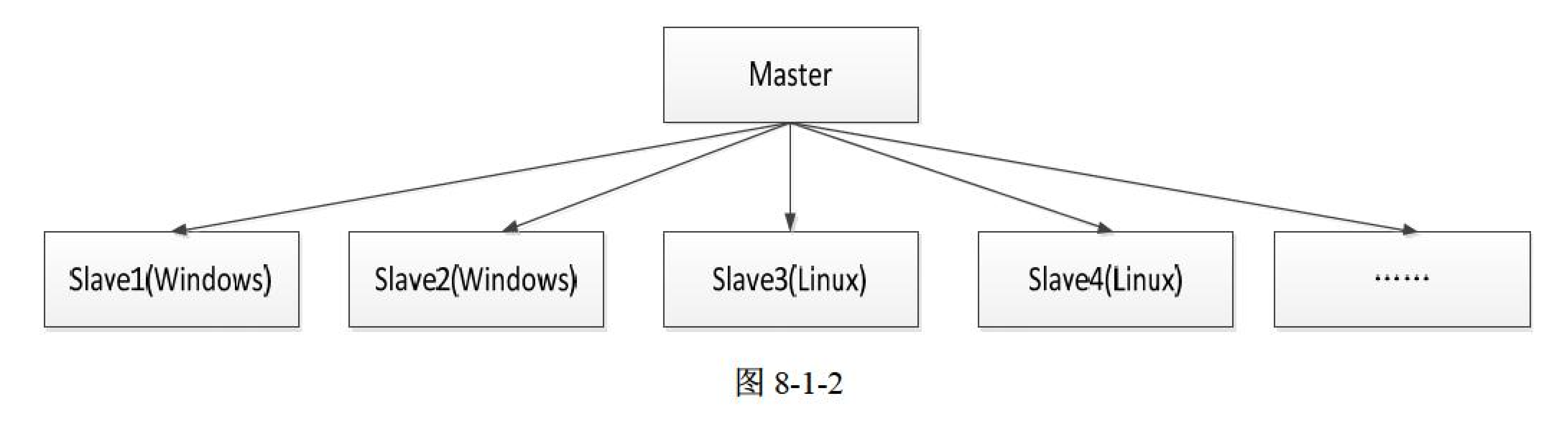


图 8-1-2

图 8-1-3 是一个 Jenkins 主从关系的架构图。用户可以通过登录到 Jenkins 的 Master 管理界面中去进行节点的配置管理、Jenkins 的插件管理、Job 任务的配置和分发、Jenkins 用户的管理、系统运行的监控以及 Jenkins 的系统全局设置等操作。Jenkins 也提供了 REST API 的方式供用户或者其他系统来调用 Jenkins。API 有 XML API、JSON API 以及 Python API 三种不同的表现类型。

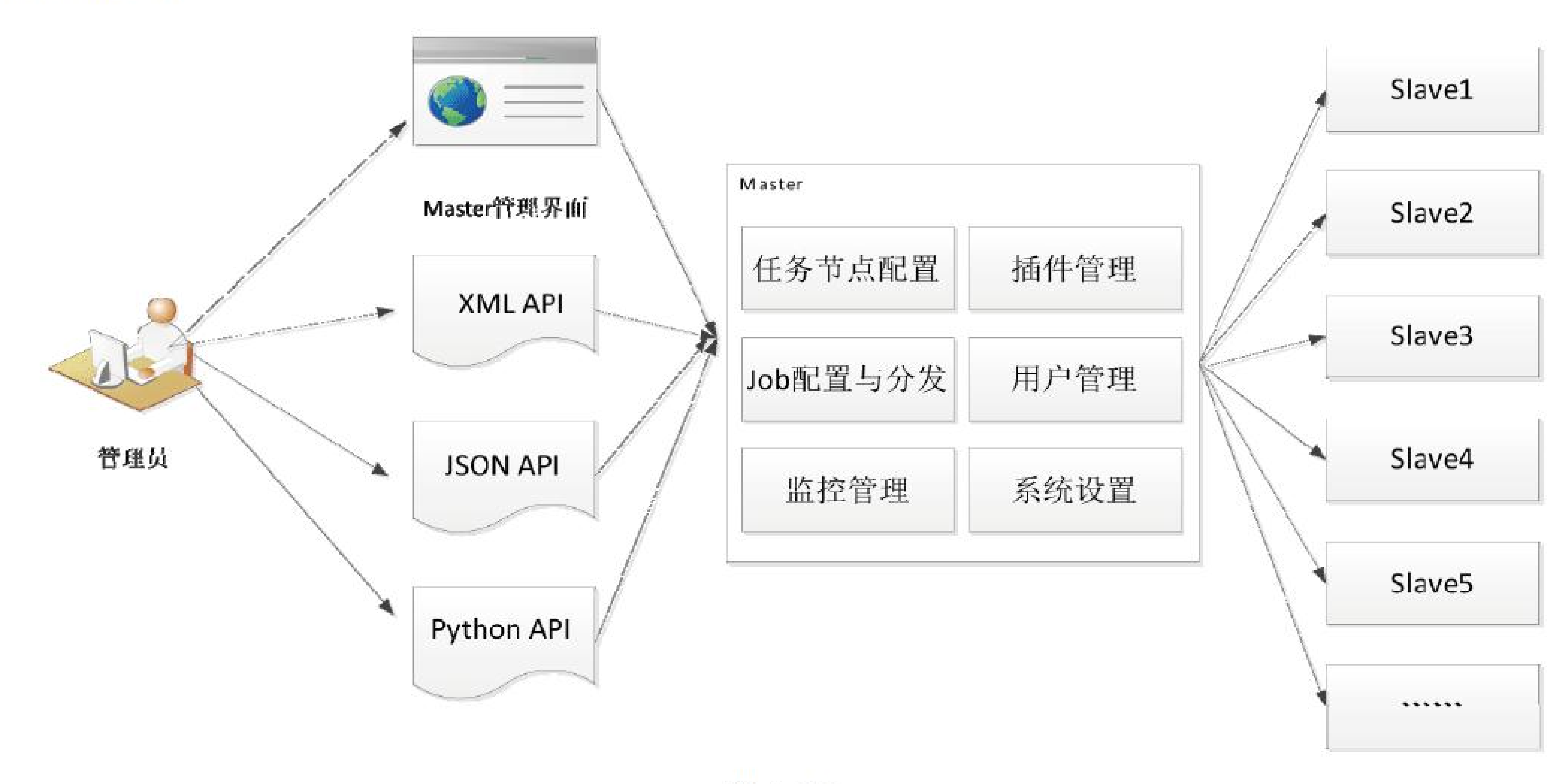


图 8-1-3

Jenkins 在安装完成并且启动成功后，可以通过访问 url 地址（http://localhost:8080/jenkins/user/admin/api/）来获取 API，如图 8-1-4 所示。

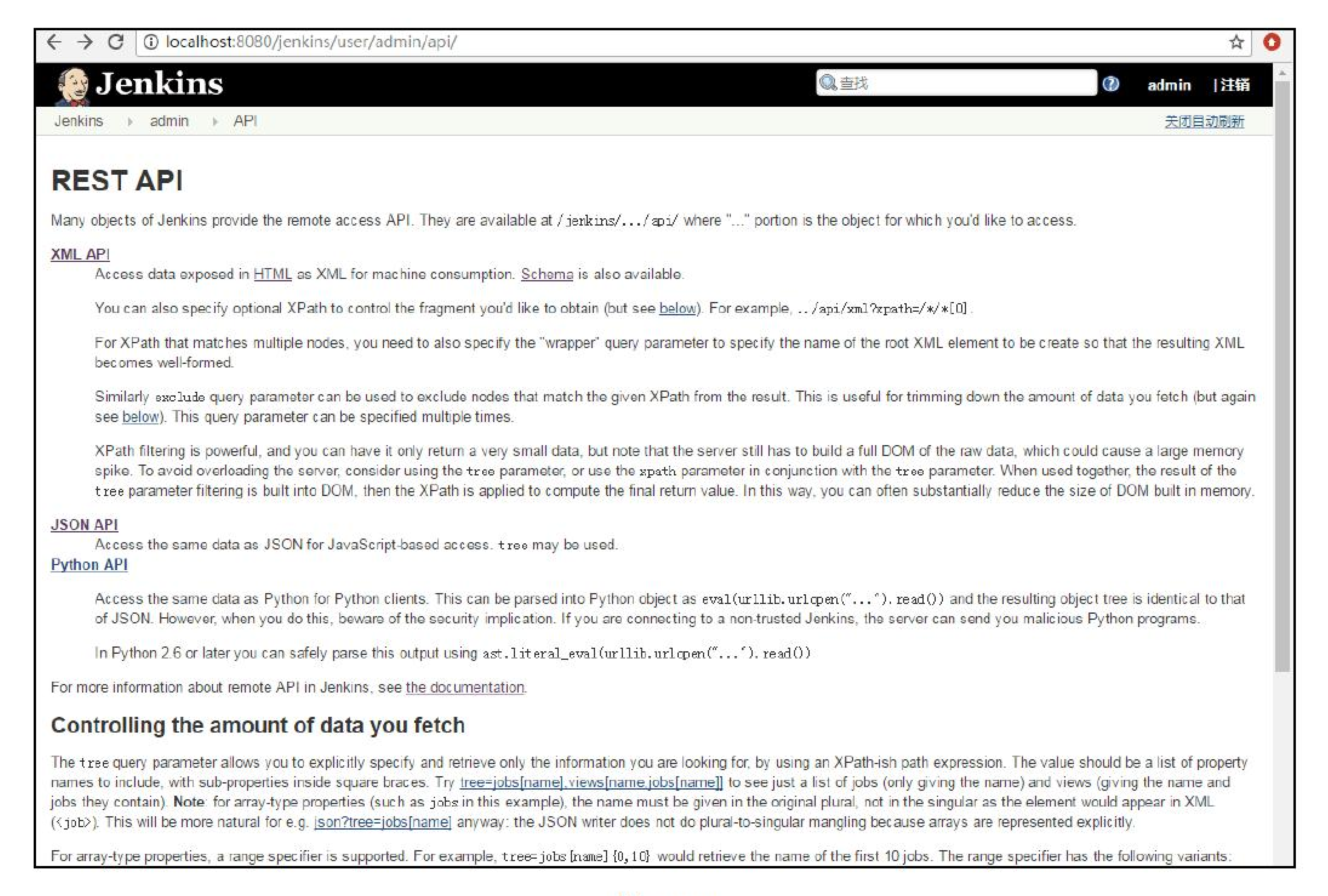


图 8-1-4

通过访问网址 https://python-jenkins.readthedocs.io/en/latest/api.html 可以获取 Python API 相关的帮助信息。

常见的 Jenkins Python API 介绍如表 8-1-1 所示。

表 8-1-1　常见的 Jenkins Python API

名称	描述
get_job_info	获取某个 Jenkins 任务的相关信息，获取到的任务信息将以 Python 字典的形式返回，相关的参数包括： ● name：任务的名称，传入的是一个 String 类型字符串 ● depth：JSON 的深度，传入的是 int 类型，默认为 0 ● fetch_all_builds：是否获取该 Jenkins 任务下的所有构建任务信息，默认为 False，传入的是一个布尔值类型参数
get_job_info_regex	这是一个通过正则表达式来模糊查询 Jenkins 任务信息的接口，获取 Jenkins 的任务信息能与指定正则表达式相匹配的相关任务信息，结果会以 Python List 的形式返回，相关的参数包括： ● pattern：String 类型的正则表达式 ● depth：JSON 的深度，传入的是 int 类型，默认为 0 ● folder_depth：搜索目录的深度，传入的也是 int 类型，默认为 0
get_job_name	获取 Jenkins 任务的名称，相关的参数包括： ● name：String 类型的任务名称

（续表）

名称	描述
debug_job_info	获取某个 job 更多的 debug 级别的详细信息，相关的参数包括： ● job_name：Jenkins 任务的名称
get_queue_item	获取 job 任务的队列信息，结果以 Python 字典的形式返回，相关的参数包括： ● number：队列的大小，传入的是 int 类型 ● depth：JSON 的深度，传入的是 int 类型，默认为 0
get_build_info	获取某个 Jenkins 任务的构建信息，结果以 Python 字典的形式返回，相关的参数包括： ● name：Jenkins 任务的名称，传入的是 String 类型的字符串 ● number：Jenkins 任务的构建编号，传入的是 int 类型 ● depth：JSON 的深度，传入的是 int 类型，默认为 0
get_build_test_report	获取某个任务构建的测试报告，结果以 Python 字典的形式返回，相关的参数包括： ● name：Jenkins 任务的名称，传入的是 String 类型的字符串 ● number：Jenkins 任务的构建编号，传入的是 int 类型 ● depth：JSON 的深度，传入的是 int 类型，默认为 0
get_build_env_vars	获取某个 Jenkins 任务构建的环境变量信息，结果以 Python 字典的形式进行返回，相关的参数包括： ● name：Jenkins 任务的名称，传入的是 String 类型的字符串 ● number：Jenkins 任务的构建编号，传入的是 int 类型 ● depth：JSON 的深度，传入的是 int 类型，默认为 0
cancel_queue	取消一个构建队列中的任务，相关的参数包括： ● id：一个待构建的 Jenkins 任务的 id 编号，传入的是 int 类型
get_plugins_info	获取 Master 节点上安装的所有插件信息，相关的参数包括： ● depth：JSON 的深度，传入的是 int 类型，默认为 2
get_all_jobs	获取所有的 Jenkins 任务，结果以 Python List 形式返回，相关的参数包括： ● folder_depth：搜索的目录层级，传入的是 int 类型，默认值为（None 代表会在所有的目录层级中进行查找）
create_job	创建一个 Jenkins 任务，相关的参数包括： ● name：待创建的 Jenkins 任务的名称 ● config_xml：待创建的 Jenkins 任务的 xml 配置文件
delete_job	删除一个 Jenkins 任务，相关的参数包括： ● name：待删除的 Jenkins 任务的名称
disable_job	关闭一个 Jenkins 任务。执行操作后，该 Jenkins 任务将不会再被执行。相关的参数包括： ● name：待关闭的 Jenkins 任务的名称
enable_job	打开一个被关闭的 Jenkins 任务。执行操作后，该 Jenkins 任务将会再次被执行，相关的参数包括： ● name：待打开的 Jenkins 任务的名称

（续表）

名称	描述
get_job_config	获取某个 Jenkins 任务的配置信息，结果以 xml 的形式返回，相关的参数包括： ● name：Jenkins 任务的名称
get_nodes	获取已经和 Master 节点建立连接的所有 Jenkins 节点, 结果以 Python List 的形式进行返回，相关的参数包括： ● depth：JSON 的深度，传入的是 int 类型，默认为 0
get_node_info	获取某个 Jenkins 节点的相关信息，结果以 Python 字典的格式进行返回，相关的参数包括： ● name：节点的名称 ● depth：JSON 的深度，传入的是 int 类型，默认为 0
create_node	创建一个 Jenkins 节点，相关的参数包括： ● name：待创建的 Jenkins 节点的名称，传入的参数为 String 类型值 ● numExecutors：该节点分配的执行器的数量，传入的参数为 int 类型值 ● nodeDescription：待创建的节点的描述信息，传入的参数为 String 类型值 ● remoteFS：远程文件系统的路径，传入的参数为 String 类型值 ● labels：给待创建节点打的标签，传入的参数为 String 类型值 ● exclusive：是否使用该节点仅仅执行固定的任务，传入的参数为布尔类型值 ● launcher：slave 节点的运行方式。Jenkins 提供了 4 种运行方式： ➢ jenkins.LAUNCHER_COMMAND。 ➢ jenkins.LAUNCHER_SSH。 ➢ jenkins.LAUNCHER_JNLP。 ➢ jenkins.LAUNCHER_WINDOWS_SERVICE。 ● launcher_params：附加的执行参数，传入的参数为 Python 字典类型
disable_node	关闭一个 Jenkins 节点，相关的参数包括： ● name：待关闭的 Jenkins 节点的名称 ● msg：关闭节点时的通知信息
enable_node	打开一个 Jenkins 节点，相关的参数包括： ● name：待打开的 Jenkins 节点的名称
delete_node	删除一个 Jenkins 节点，相关的参数包括： ● name：待删除的 Jenkins 节点的名称

Jenkins 的 Python API 中还提供了很多和 Jenkins 操作相关的接口方法，更多的内容可以参考 https://python-jenkins.readthedocs.io/en/latest/api.html#jenkins.Jenkins。

我们来看一个 Python API 的运行示例。下面这段 Python 脚本代码首先引入 Jenkins 这个 package，然后连接到本地已经启动的 Jenkins 服务器上，并且打印出该 Jenkins 服务器上的 job 总数以及获取的所有 Jenkins job 任务。

```
import jenkins

server = jenkins.Jenkins('http://localhost:8080/jenkins/', username='admin',
password='admin')
print server.jobs_count()
print server.get_all_jobs(0)
```

运行结果如图 8-1-5 所示。

```
Run JenkinsExample
"C:\Program Files (x86)\python\python.exe" F:/project/robot-framework/com/robot/remote/example/JenkinsExample.py
1
[{u'url': u'http://localhost:8080/jenkins/job/robotframework/', u'color': u'notbuilt', u'fullname': u'robotframework', u'_class': u'hudson.model.FreeStyleProject', u'name': u'robotframework'}]

Process finished with exit code 0
```

图 8-1-5

一般我们在配置一个 Jenkins 任务时会包含图 8-1-6 的常见步骤。Jenkins 任务的执行需要依赖很多不同的插件，Jenkins 的一大特性就是提供了非常丰富的插件。

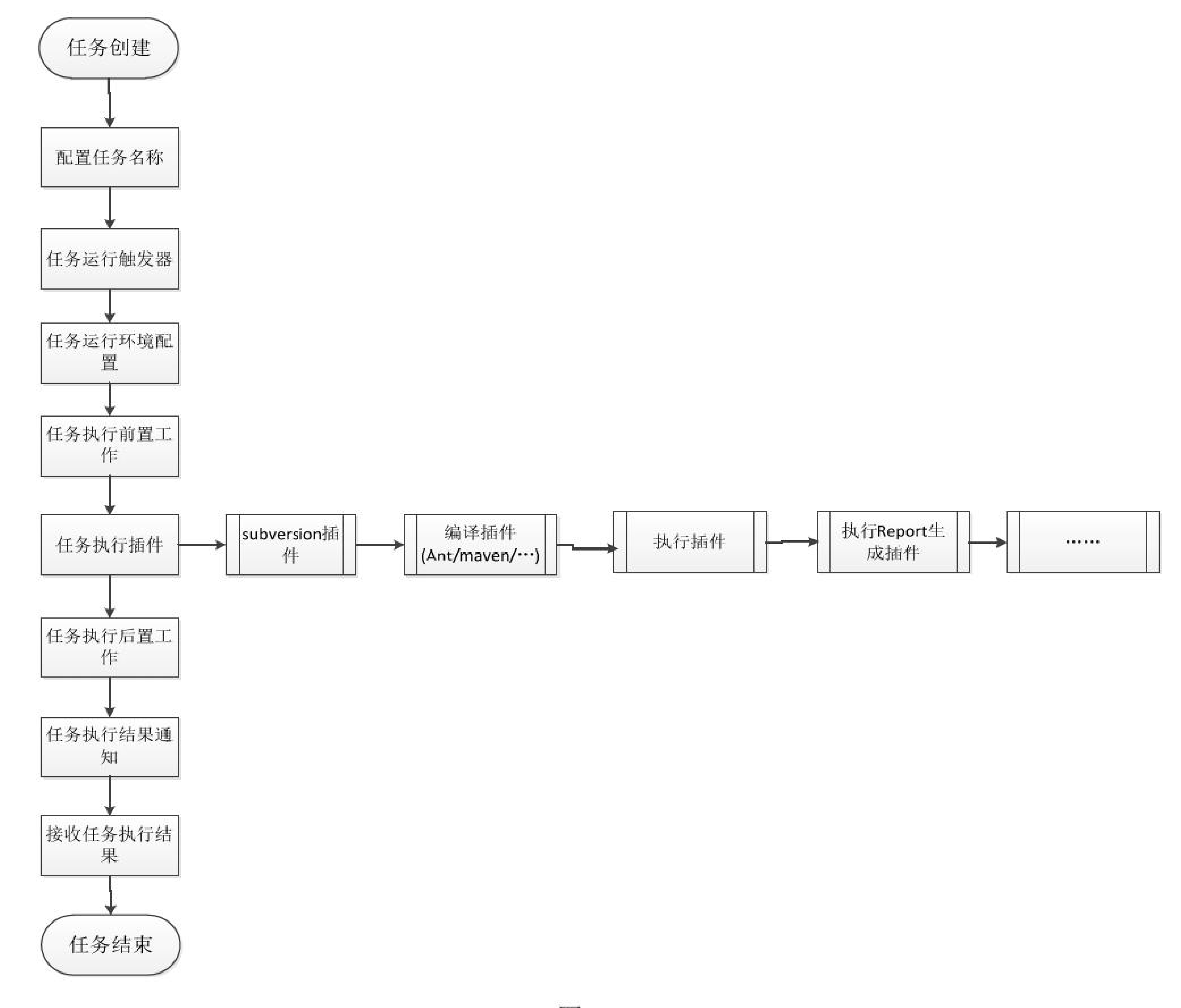


图 8-1-6

8.1.2　在 Jenkins 上运行 Robot Framework 自动化测试用例

我们首先需要建立一个 Robot Framework 的构建任务，并在任务中配置重试次数、构建的依赖日志等，如图 8-1-7 所示。

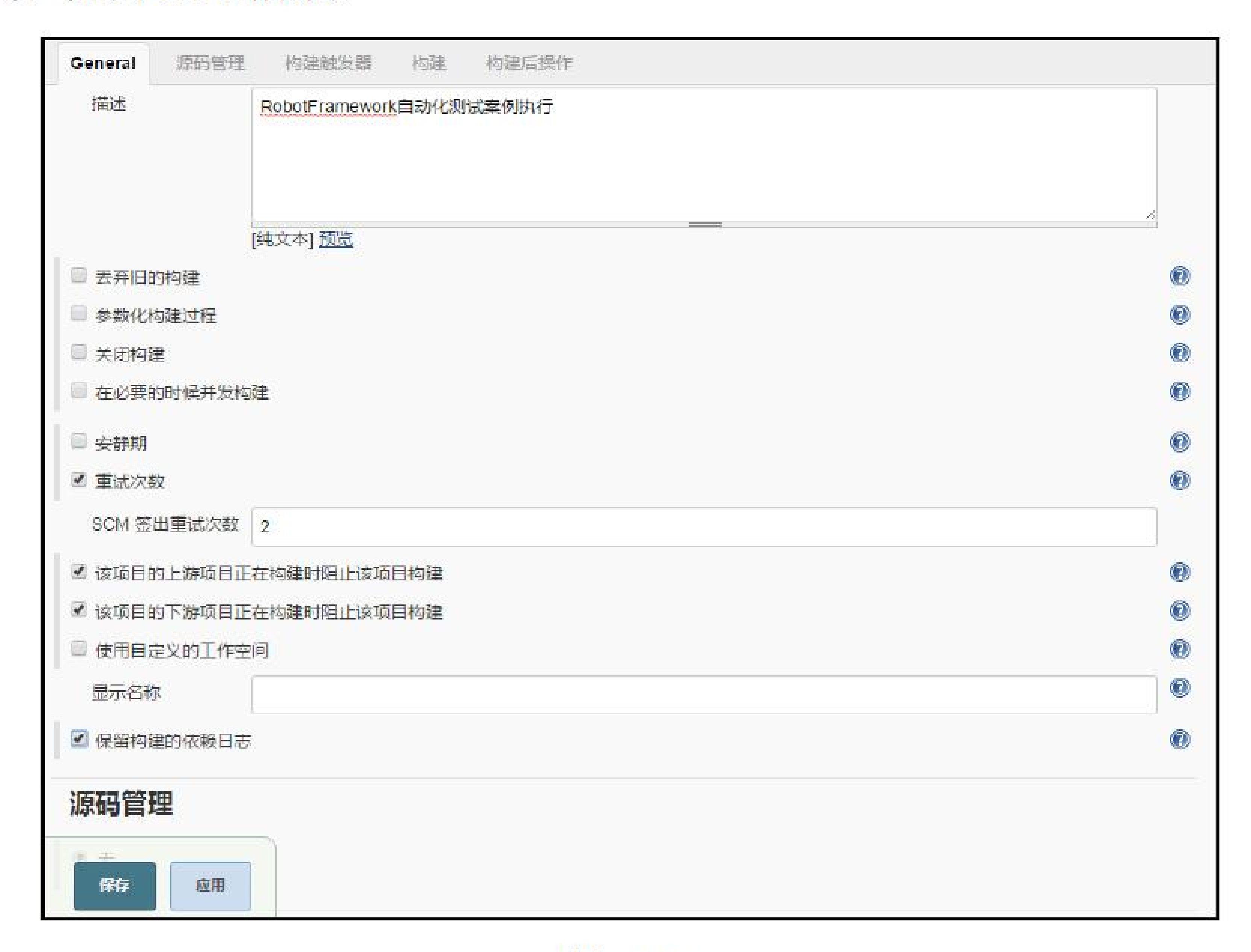


图 8-1-7

然后配置自己的源码管理。我们可以在这个步骤中配置测试用例的路径等，如图 8-1-8 所示。



图 8-1-8

接着配置构建触发器。顾名思义，触发器就是配置任务触发的方式、触发时间。Jenkins

支持触发远程构建、其他工程构建后触发、定时构建、轮询 SCM 这几种构建触发方式，如图 8-1-9 所示。



图 8-1-9

构建触发器步骤中相关输入参数的描述如表 8-1-2 所示。

表 8-1-2　构建触发器步骤中相关输入参数

选项	说明
触发远程构建	通过远程调用的方式，支持 JENKINS_URL/job/robotframework/build?token=TOKEN_NAME 或者 /buildWithParameters?token=TOKEN_NAME 两种方式，在调用时需要用身份验证令牌 token 来进行身份验证
其他工程构建后触发	其他的任务在构建完成后就会自动触发当前的任务运行，可以指定只有其他任务运行成功后才能触发当前任务，也可以指定其他任务构建失败时触发当前任务
定时构建	可以指定通过定时任务的形式来触发构建。触发的形式（和我们见过的很多定时任务触发器很像）如下： MINUTE HOUR DOM MONTH DOW ● MINUTE：表示分钟，取值为 0~59。若其他值不做设定，则表示每个设定的分钟都会构建。例如 5 * * * * ，表示每小时的第 5 分钟都会构建一次 ● HOUR：表示小时，取值为 0~23。若其他值不做设定，则表示每个设定小时的每分钟都会构建。例如 * 5 * * * ，表示在每天 5 点的时候，在一小时内每一分钟构建一次 ● DOM：表示一个月的第几天，取值为 1~31。若其他值不做设定，则表示每个月的那一天每分钟都会构建一次。例如 * * 5 * *，表示在每个月 5 号的时候，从 0 点开始每分钟构建一次

	● MONTH：表示第几月，取值为 1~12。若其他值不做设定，则表示每年的那个月每分钟都会构建一次，例如 * * * 5 *，表示在每年的 5 月份，从 1 号 0 点开始每分钟构建一次 ● DOW：表示一周中的第几天，取值为 0~7，其中 0 和 7 代表的都是周日。若其他值不做设定，则表示每周的那一天每分钟构建一次。例如 * * * * 5，表示每周五从 0 点开始每分钟构建一次
轮询 SCM	表示配置任务轮询来执行，配置的方式和定时构建非常相似

在配置完构建触发器后，接着可以配置构建步骤，也就是自动化测试用例的执行步骤，如图 8-1-10 所示，构建步骤可以配置多个，每一个构建步骤都可以通过拖动的方式来调整其执行的先后顺序。

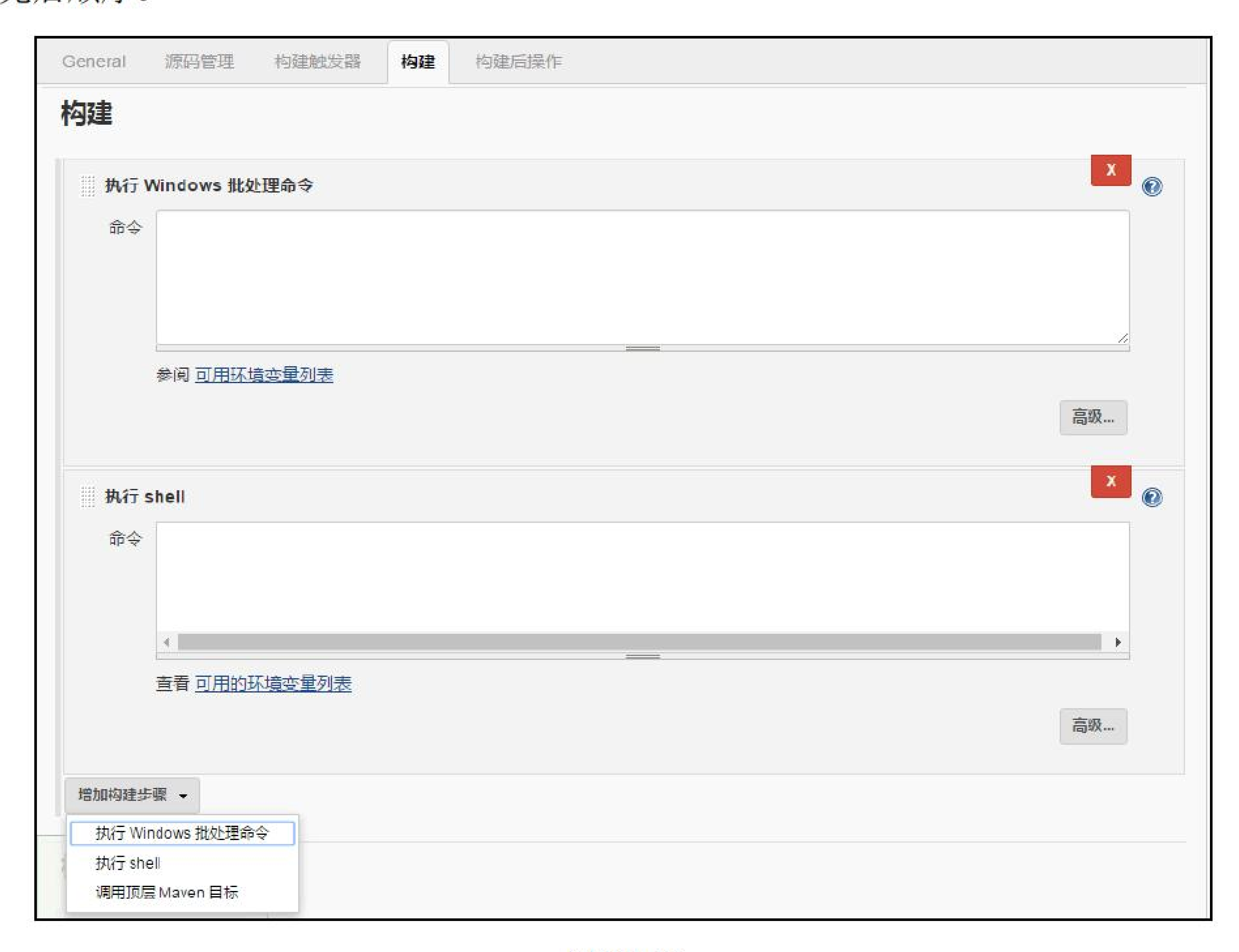


图 8-1-10

表 8-1-3 中描述了两种不同构建方式的区别。

表 8-1-3　不同构建方式的区别

选项	说明
执行 Windows 批处理命令	一般用于 Windows 环境上执行，可以是一个 Windows 的批处理命令或者是调用一个 Windows 的批处理脚本
执行 shell	一般用于 Linux 环境上执行，可以是一条 Linux 的操作命令或者是调用一个 Linux 上的 shell 脚本

在这里配置一个 Windows 的批处理命令来执行我们从 RIDE 中已经编写好的 Robot

Framework 的自动化测试用例：pybot.bat --argumentfile c:\users\yongqing\appdata\local\temp\RIDEbbpu6m.d\argfile.txt --listener "C:\Program Files (x86)\python\lib\site-packages\robotide\contrib\testrunner\TestRunnerAgent.py:32451:False" F:\project\RobotFrameworkTest1\RobotFrameworkTest1，如图 8-1-11 所示。



图 8-1-11

最后构建任务结束后的操作，包括归档、继续构建其他的工程任务、记录指纹跟踪等，如图 8-1-12 所示。

图 8-1-12

在任务创建完成后，我们就可以构建已经配置好的自动化测试用例任务了，如图 8-1-13 所示。



图 8-1-13

在任务执行完成后，我们可以查看自动化测试用例任务的构建执行日志，如图 8-1-14 所示。

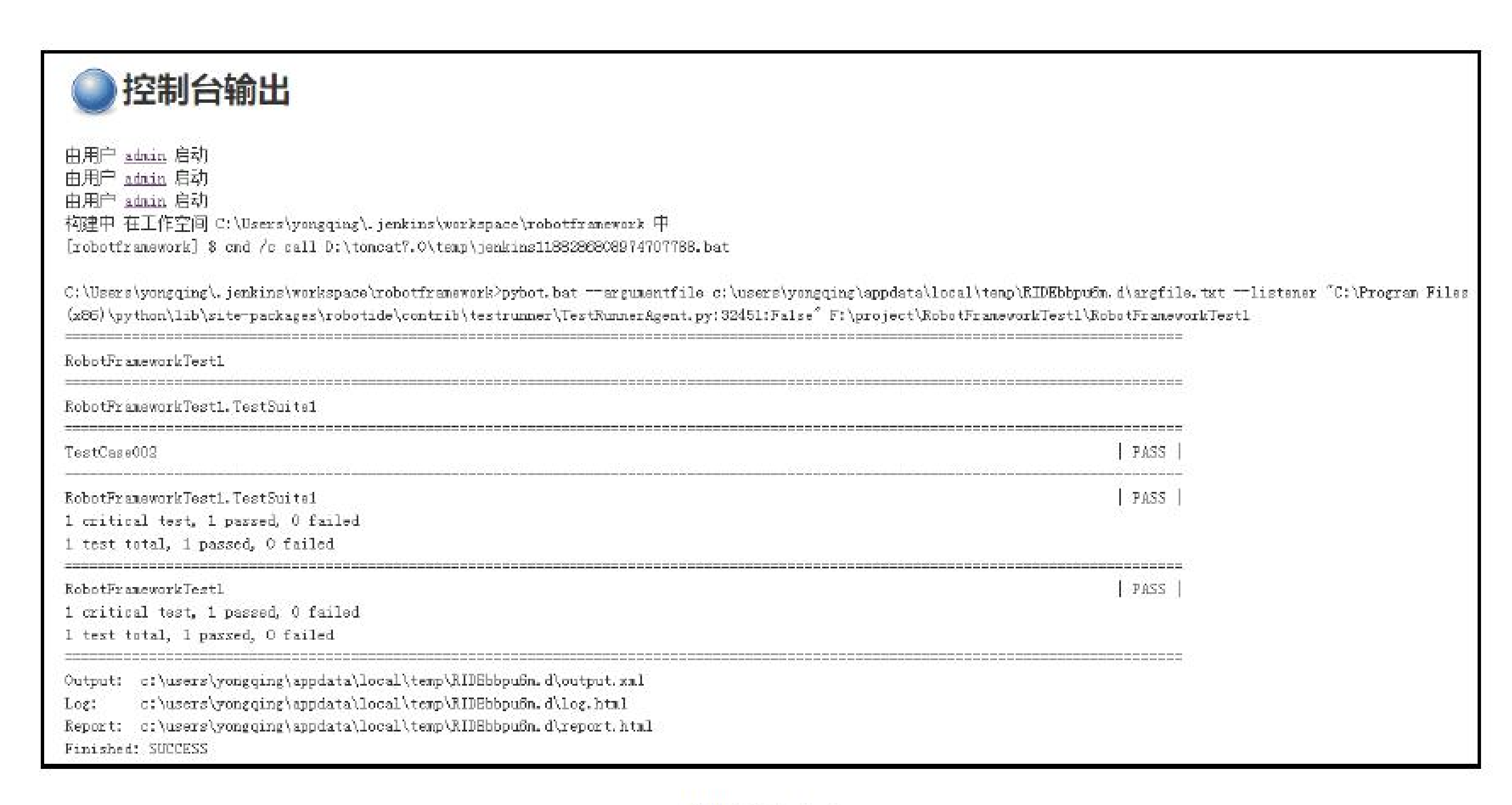


图 8-1-14

自动化测试用例执行完成后，我们也可以查看到测试用例 RobotFramework 为我们自动生成的自动化测试用例执行报告，如图 8-1-15 和图 8-1-16 所示。

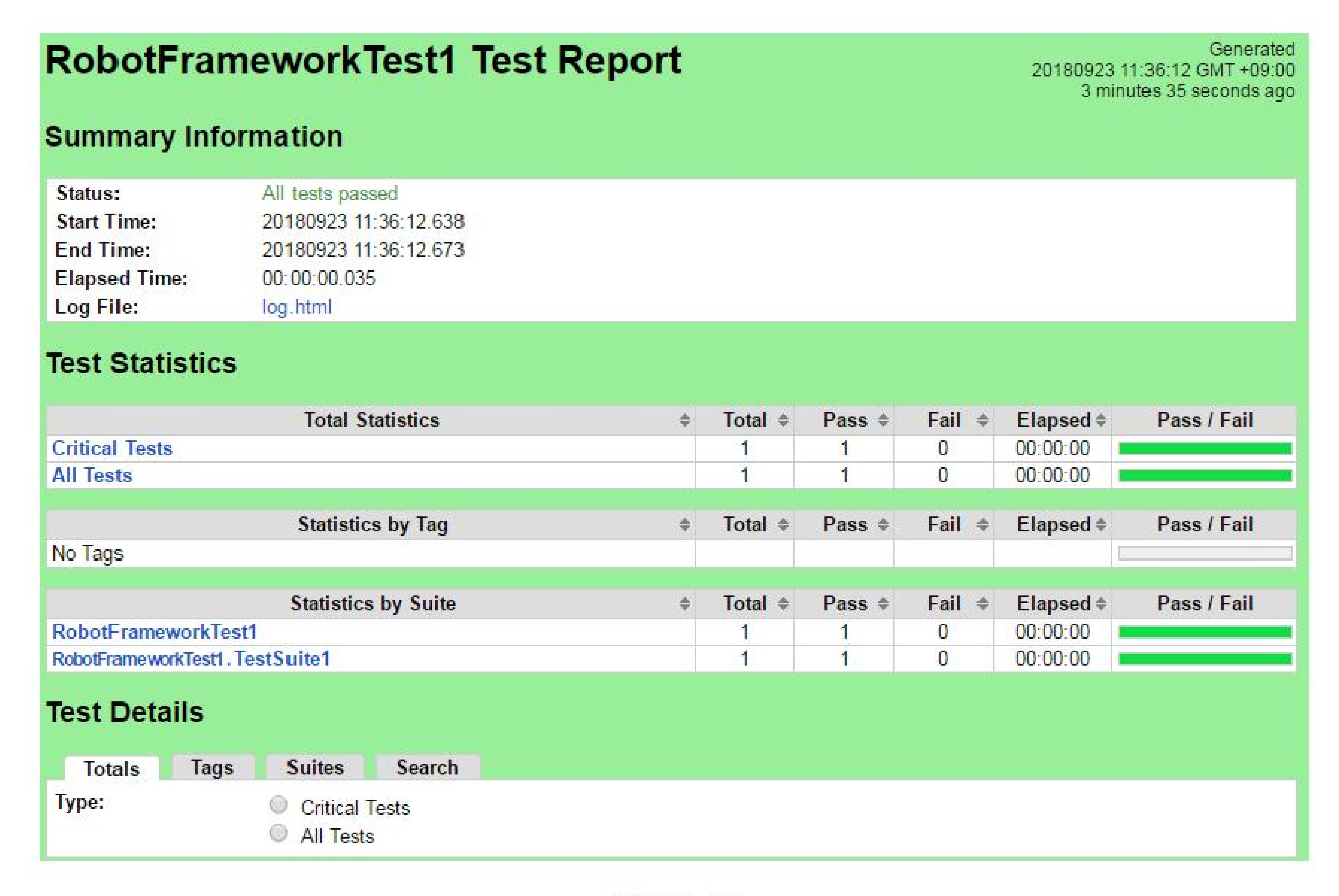


图 8-1-15

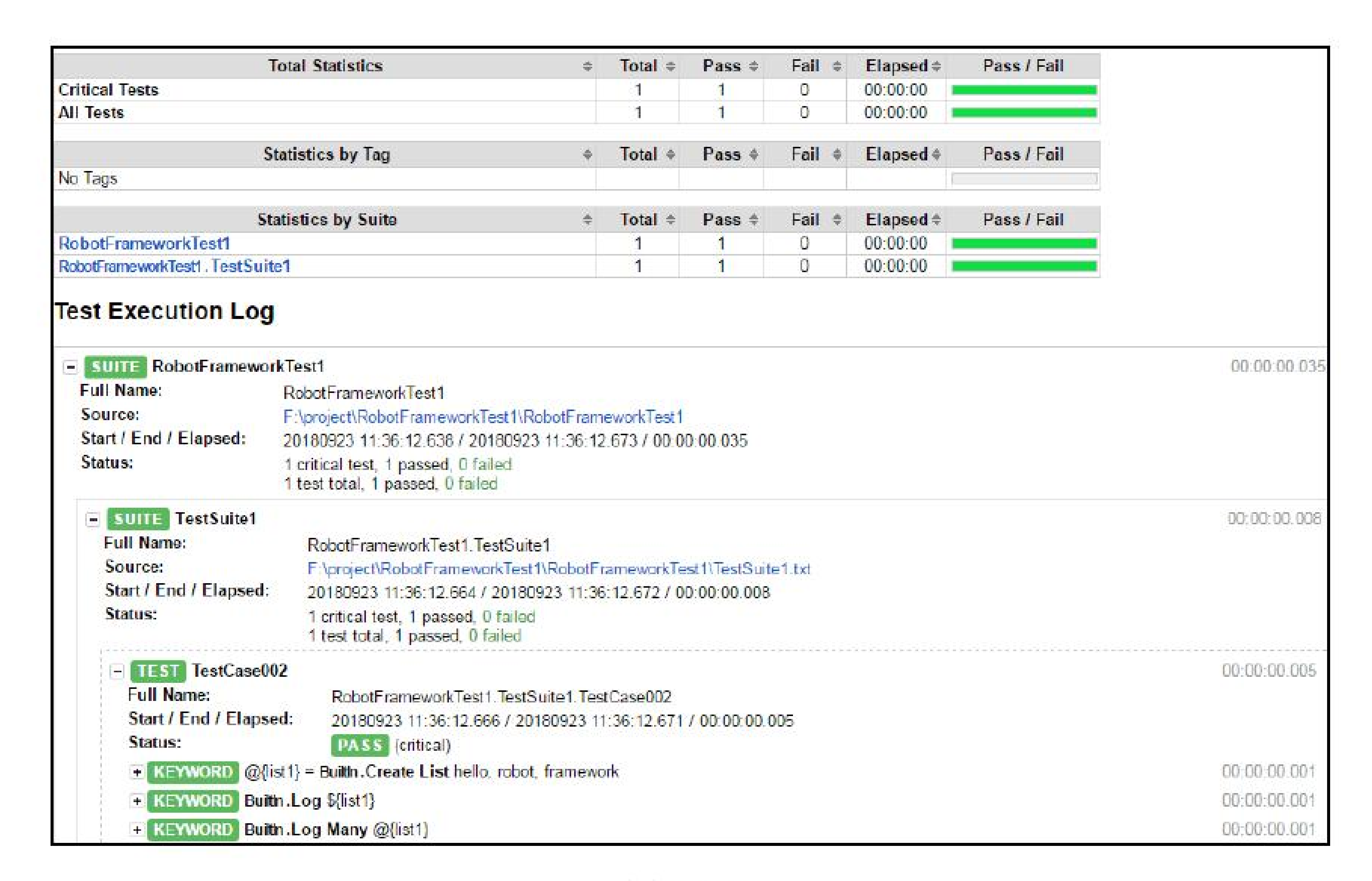


图 8-1-16

8.2 如何做好自动化测试平台框架的设计

自动化测试如果需要能高效快速地支撑软件项目的测试、项目的快速迭代以及上线，除了以上我们介绍的需要许多的 Lib 来支持以及需要高效地去编写自动化测试用例外，还需要一个

好的自动化测试框架平台来支撑我们的自动化测试用例的执行以及分析。自动化测试平台一般需要解决测试用例执行任务的创建、任务的管理、任务的调度、测试用例的解析，测试用例分别根据不同的测试类型分发给不同的测试工具或者测试 Lib 库进行执行，并且需要对执行的数据进行分析，得出质量数据，然后才好给出对应的测试报告分析数据给测试经理，由项目经理来辅助持续地改善项目版本的质量。图 8-2-1 是一个自动化测试平台框架的架构设计图。

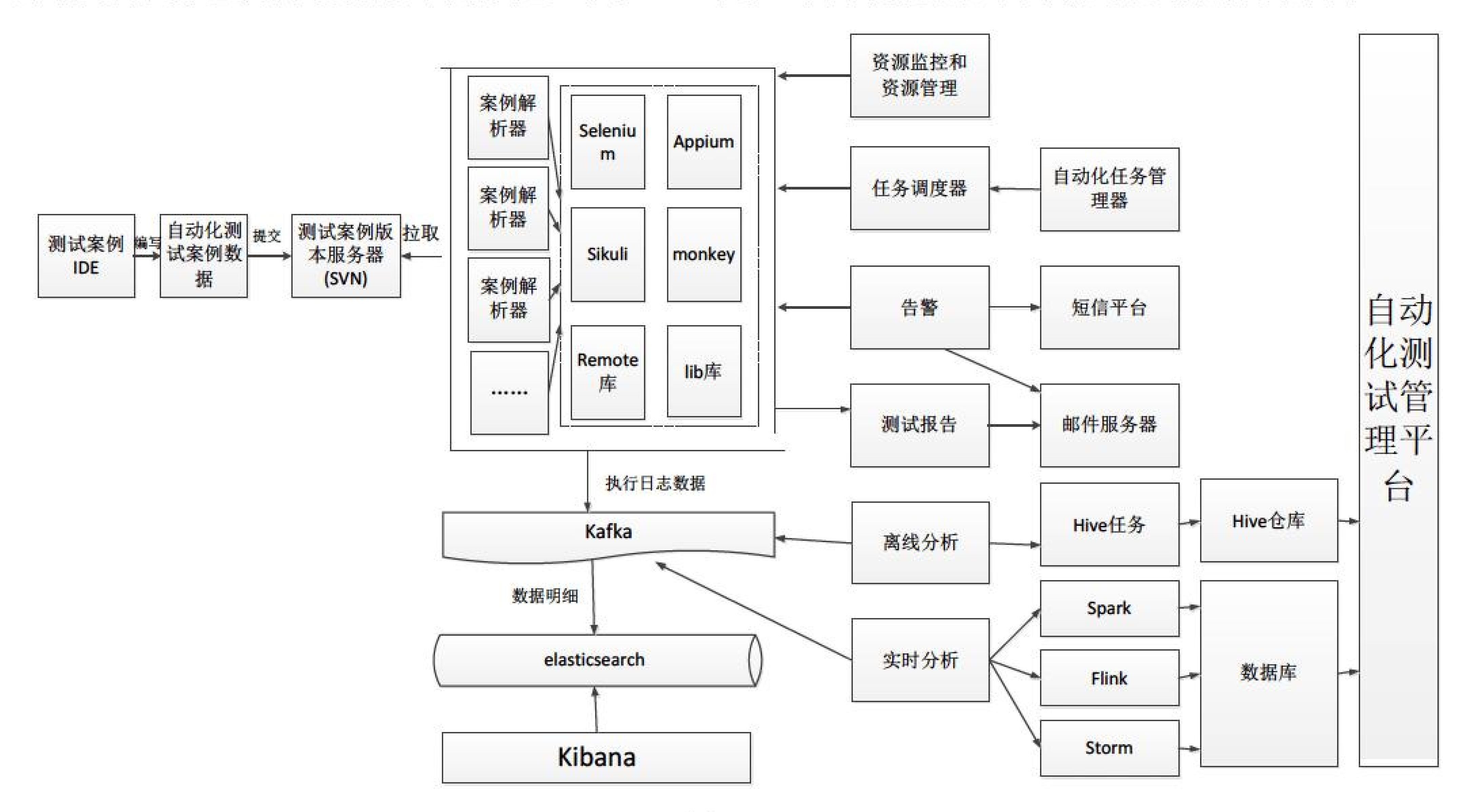


图 8-2-1

框架图中的自动化测试平台框架主要包含 4 部分：自动化测试用例编写的 IDE 工具，自动化测试用例的执行，测试任务的管理和监控，测试用例执行后的数据分析。

一个通用的自动化测试用例编写的 IDE 工具一般需要包含的功能如图 8-2-2 所示。

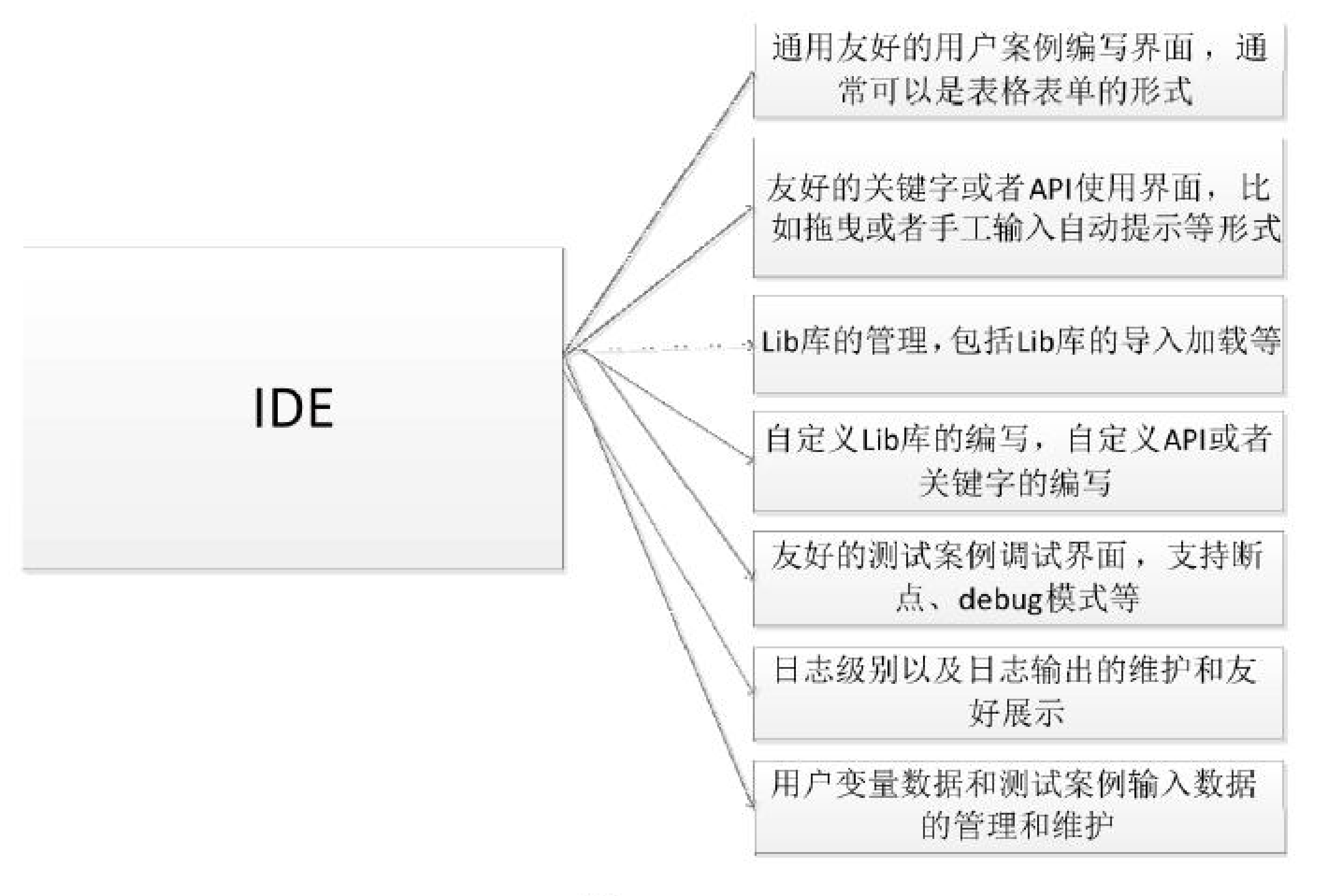


图 8-2-2

友好的用户案例编写界面一般可以用表格或者表单的形式来完成，比如 Robot Framework 就是典型的表格界面，也可以用 Eclipse 的插件式开发实现这一需求。

在平台中，一个自动化测试用例的执行一般包括如图 8-2-3 所示的基本步骤，最核心的就是将解析的案例数据分发给不同的 Lib 库去执行。

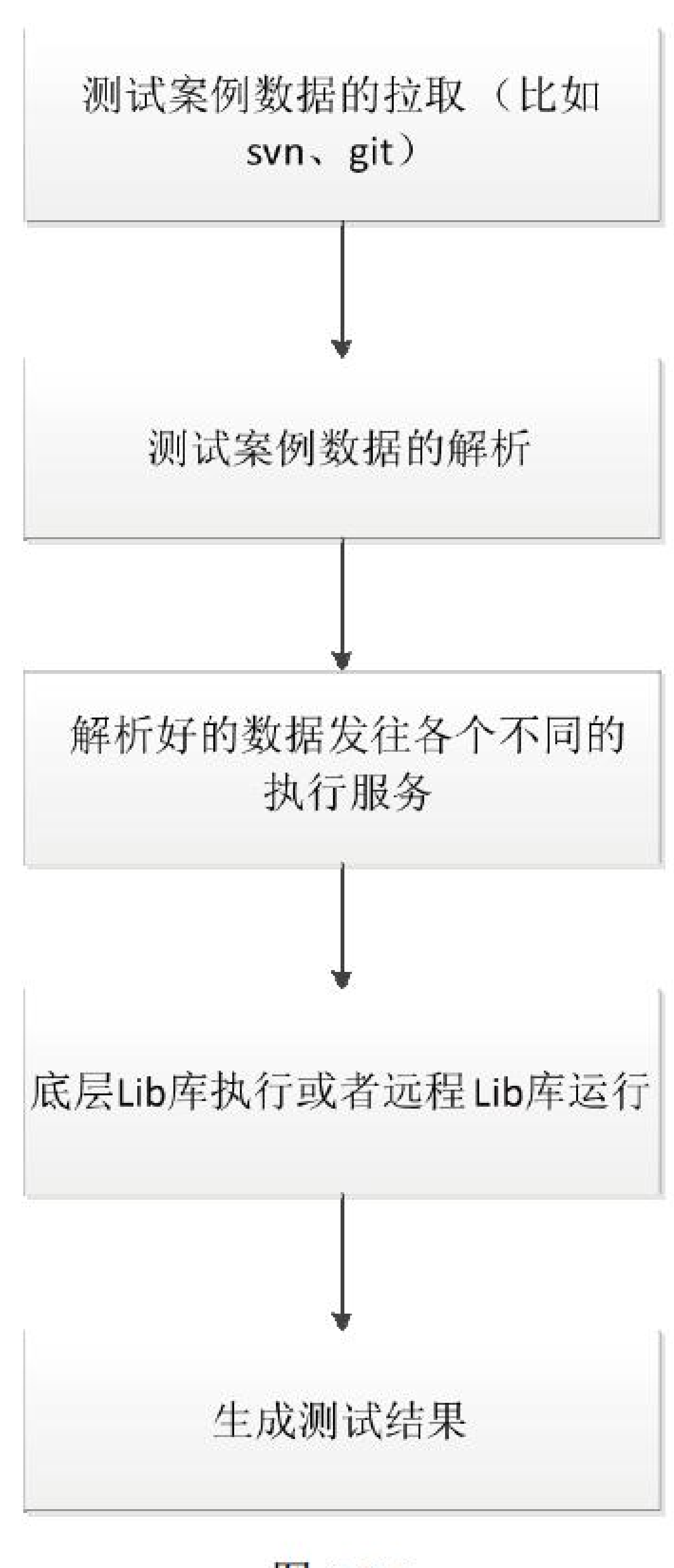


图 8-2-3

在做数据分析时，可以借用很多现在主流的大数据组件，如图 8-2-4 所示。数据分析通常包括实时数据分析和离线数据分析，在分析中我们还会用到很多消息队列中间件，比如 Kafka。数据明细可以采用 Elasticsearch 来进行存储，这和 Elasticsearch 本身的特点有关系，可以支持大数据的存储，方便以后我们来做全文检索。

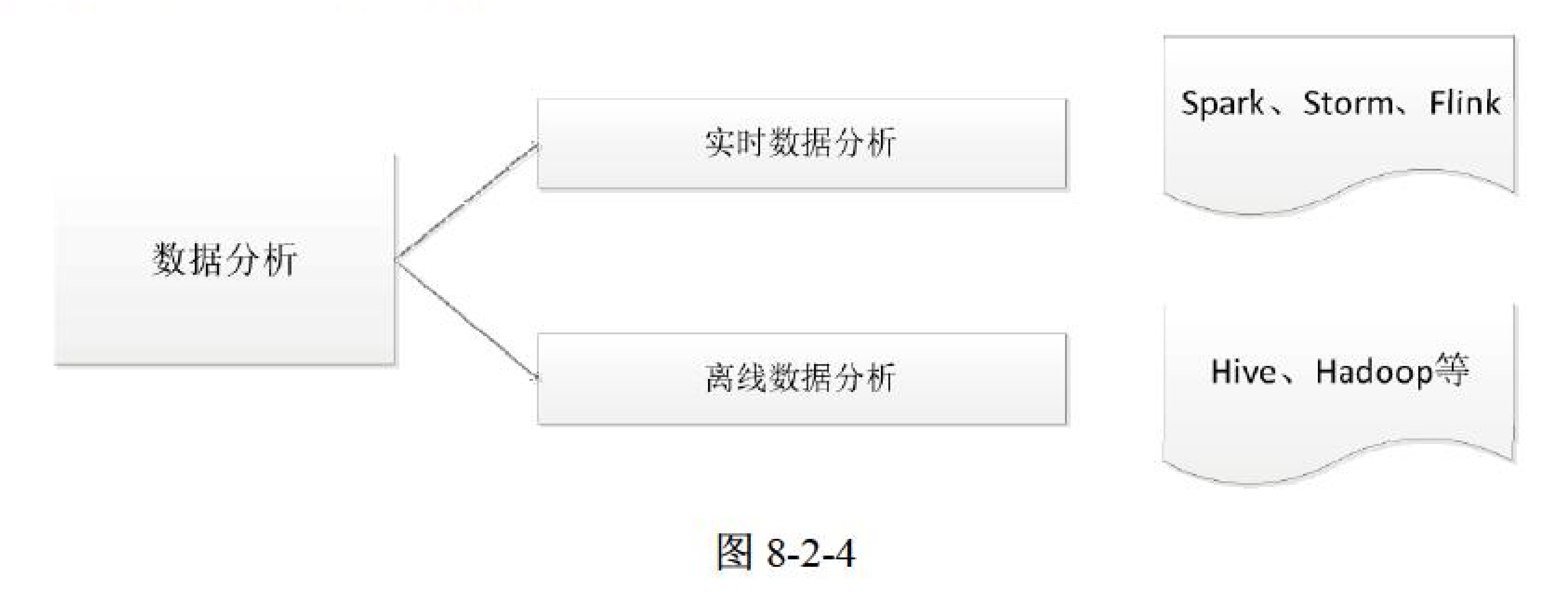


图 8-2-4

平台框架图中包含的组件如表 8-2-1 所示。

表 8-2-1　平台框架图中包含的组件

组件	组件说明
IDE	RIDE、Eclipse 插件、IDEA 插件
测试用例版本服务器	Svn、git 等版本控制服务器
自动化测试用例数据	自动化测试用例文件
案例解析器	负责自动化测试用例文件数据的读取和解析，翻译成自动化测试工具或者自动化测试 Lib 库可以执行的命令
资源监控和资源管理	负责各个执行器、案例解析器以及自动化工具服务器的资源监控和资源管理
任务调度器	调度自动化测试执行任务在多个节点上的执行
自动化任务管理器	接受自动化任务的提交、自动化任务的管理（任务启用、任务停用、任务新增、修改、删除等）
告警	任务执行异常、超时等，自动产生告警
短信平台	负责告警短信的发送以及短信发送规则的配置
邮件服务器	接收生成的测试报告或者告警信息，然后以邮件形式发送给对应的相关人员
Kafka	执行日志数据的缓存队列存储
Elasticsearch	测试用例执行的日志明细数据存储，支持全文搜索
Kibana	一个开源的分析和可视化平台，旨在与 Elasticsearch 进行结合。Kibana 提供了搜索、查看和与存储在 Elasticsearch 索引中的数据进行交互的功能。开发者或运维人员可以轻松地执行高级数据分析，并在各种图表、表格和地图中可视化数据
离线分析	负责对日志定时做离线分析处理，比如通过 Hive 等大数据方式进行数据归类分析
实时分析	将日志数据，通过大数据实时数据流的分析，提取案例执行过程的相关日志数据做实时的分析和统计，常用的实时流分析工具有 Spark、Storm、Flink 等

图 8-2-5 是自动化平台框架设计的一个分层结构图。



图 8-2-5

我们对常见的自动化测试架构思想做一个对比，如表 8-2-2 所示。

表 8-2-2 常见的自动化测试架构思想的对比

名称	说明
数据驱动测试	数据驱动测试的思想是将我们的自动化测试脚本和测试数据放在共同的测试架构中，提供可重用的测试逻辑。这样做的目的是减少测试维护的工作量以及便于改善测试用例的覆盖率。测试用例需要输入的测试数据和测试完成后的测试结果数据都会被存储在同一个数据库或者数据源中，并且将测试的数据和测试逻辑分开。这样测试数据发生了变化时不会影响到我们的测试逻辑，并且同一套测试逻辑可以针对多种数据来进行测试，尽量提高测试逻辑的使用效率和复用效率
模块驱动测试	模块驱动测试的思想是使用独立的脚本或者代码来对应每一个待测试的模块单元和功能。模块驱动测试引入的是编程语言中的面向对象编程中的抽象和模块独立封装的思想，即将测试代码和每一个测试模块进行解耦，减低自动化测试脚本或者自动化测试代码的维护成本，同时增强可扩展性。测试的执行者不需要知道单元模块的内部实现，只需要调用单元模块对外提供的抽象接口方法即可，单元模块的功能需求发生变化时，只需要修改该单元模块的内部实现，对外提供的抽象接口方法依然可以做到不发生变化
关键字驱动测试	Robot Framework 就是一种典型的关键字驱动测试的框架模式。关键字驱动测试通常也被认为是表格驱动测试，通过在表格中调用关键字来实现自动化测试。这种设计思想一般会将自动化测试拆分为设计和实现两个不同的阶段。RedwoodHQ 自动化测试工具框架也是靠这种思想来实现的，设计时需要尽量考虑关键字的通俗易懂以及通用性，也就是可以在不同的测试用例或者场景中高效地复用。关键字驱动测试的优点在于自动化测试用例的编写者不需要对脚本语言有非常深入的了解就可以完成自动化测试用例的编写
混合自动化测试	混合自动化测试是上面几种自动化测试思想的综合使用。关键字驱动测试和模块驱动测试在很多情况下就可以完美地结合起来使用，比如我们可以使用 Robot Framework 提供的自定义用户关键字来对单元模块业务进行封装，封装完成后再提供一个新的用户自定义关键字。新的用户自定义关键字就可以认为是一个抽象的接口
基于模型测试	一般基于模型测试的思想用得较少，只能适合于特定的也是基于这种模型设计思想的系统。通常情况下，这一测试模型是全部或者部分从待测试的软件系统的功能模型中提取出来的。在测试模型中描述了待测试系统的抽象行为，因此从测试模型中也可以派生出功能测试用例
行为驱动开发	行为驱动开发是一种敏捷开发的思想，使用简单的、特定于领域的脚本语言（DSL）将结构化自然语言语句转换为通俗易懂的可执行测试。行为驱动开发的根基是一种“通用语言”，通俗易懂，同时被客户和开发者用来定义系统的行为。Cucumber 就是一种行为驱动开发的自动化测试工具

8.3 其他常用的自动化测试框架介绍

8.3.1 RedwoodHQ 介绍

RedwoodHQ 与 Robot Framework 有所不同，它可以提供一个网站界面，允许多个测试团队中的多个人员一起协作，支持用 Java、Groovy、Python 和 C#等热门的编程语言来实现自动化测试脚本。RedwoodHQ 提供的 IDE 是 Web 的形式，但是也是基于关键字的方式来进行操作。要创建一个测试脚本，只需要找到要执行的操作，将其拖动到测试用例中，然后输入其期望的参数值。RedwoodHQ 在 GitHub 上的开源代码地址是 https://github.com/dmolchanenko/RedwoodHQ，如图 8-3-1 所示。

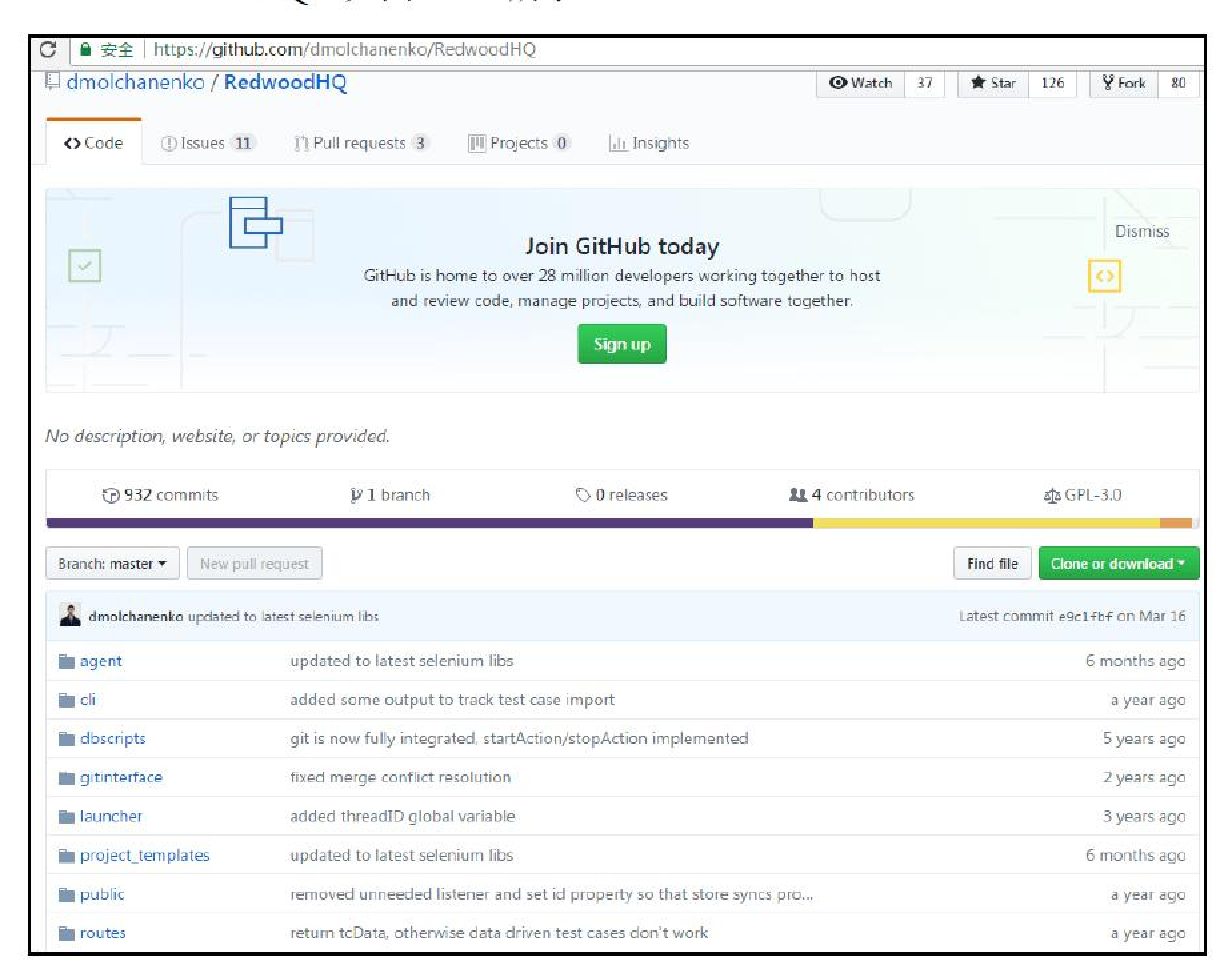


图 8-3-1

通过访问官方的网址 http://redwoodhq.com/ 可以下载 RedwoodHQ 进行安装和使用，如图 8-3-2 所示。

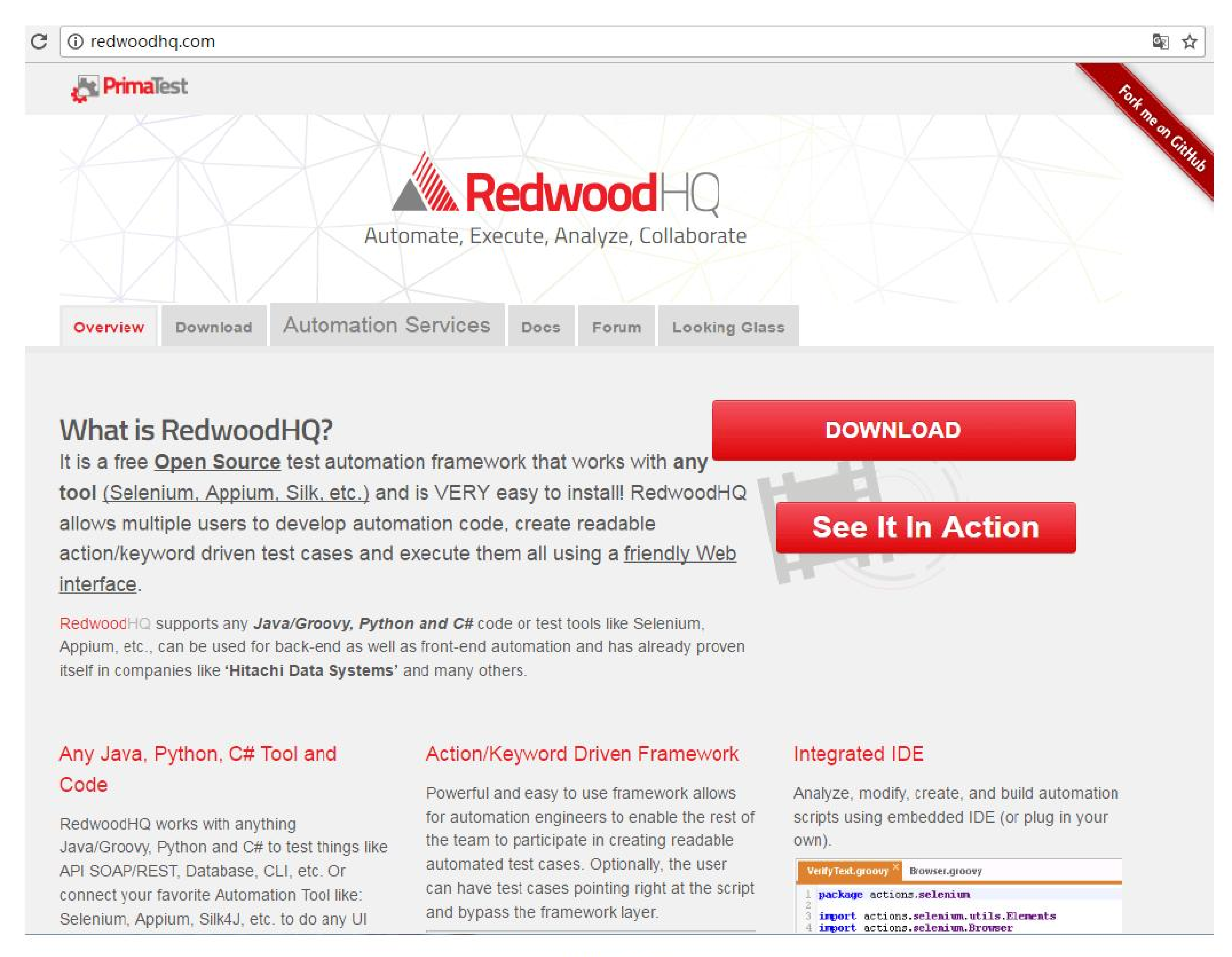


图 8-3-2

RedwoodHQ 可以支持 SOAP、REST 等服务，也支持对常见数据库的操作，还支持连接到常用的自动化测试工具，比如 Selenium、Appium、Silk4J 等。

在 RedwoodHQ 的 Web IDE 提供了 Execution（案例执行）、Test Cases（案例编写）、Actions（关键字）、Scripts（测试脚本）、Settings（IDE 设置）5 个功能菜单，如图 8-3-3 所示。

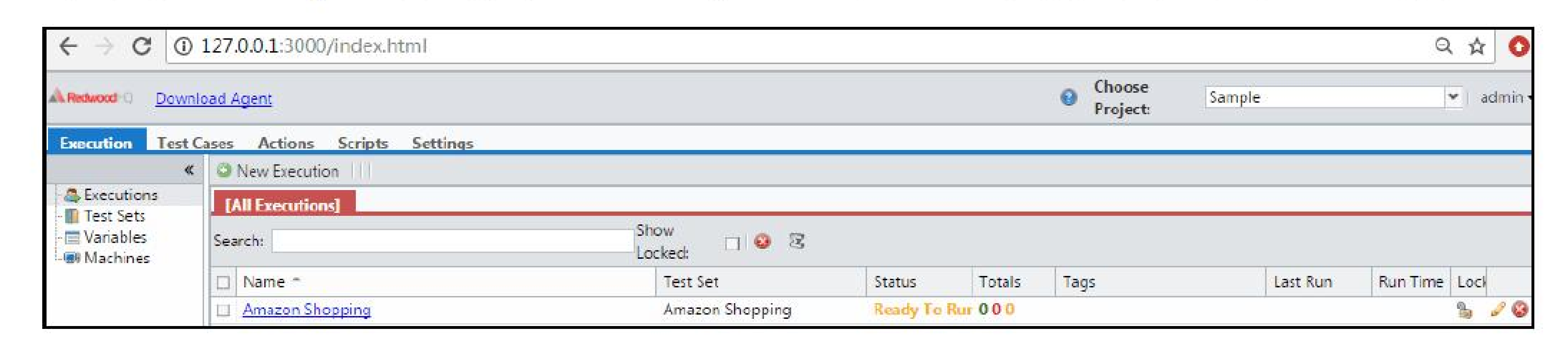


图 8-3-3

Test Case 菜单下提供了测试用例的编写功能，如图 8-3-4 所示。

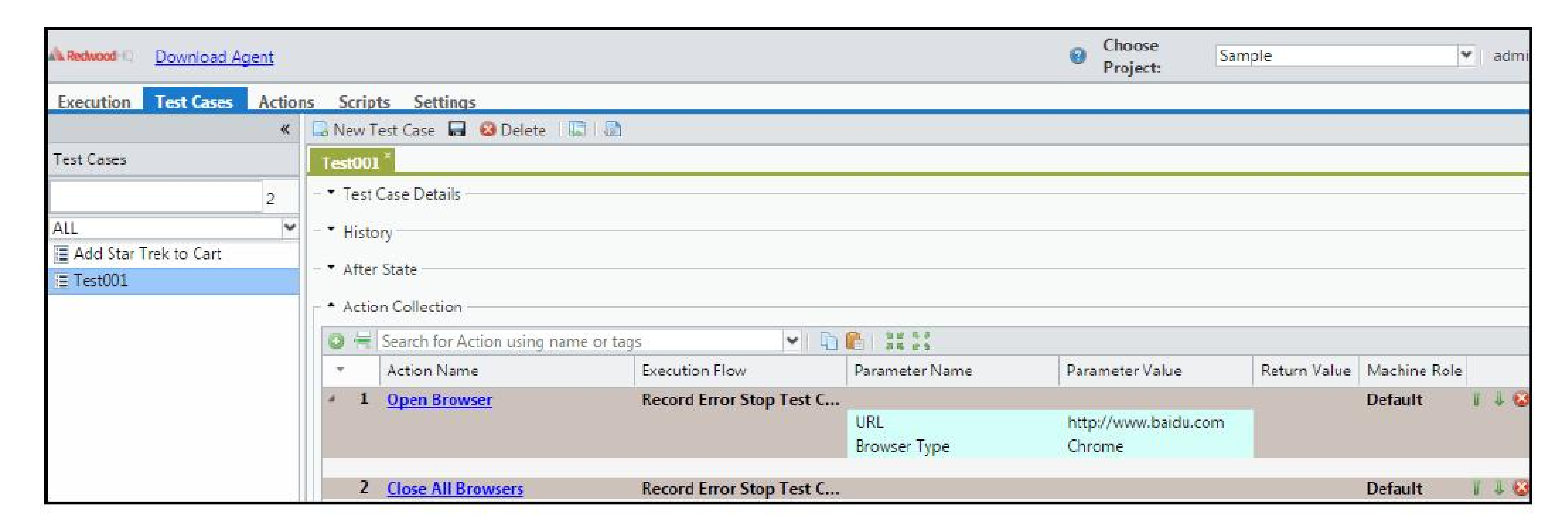


图 8-3-4

通过单击 New Test Case 按钮，可以新建一个自动化测试用例，并且可以在 Action 中选择可用的关键字操作，如图 8-3-5 所示。

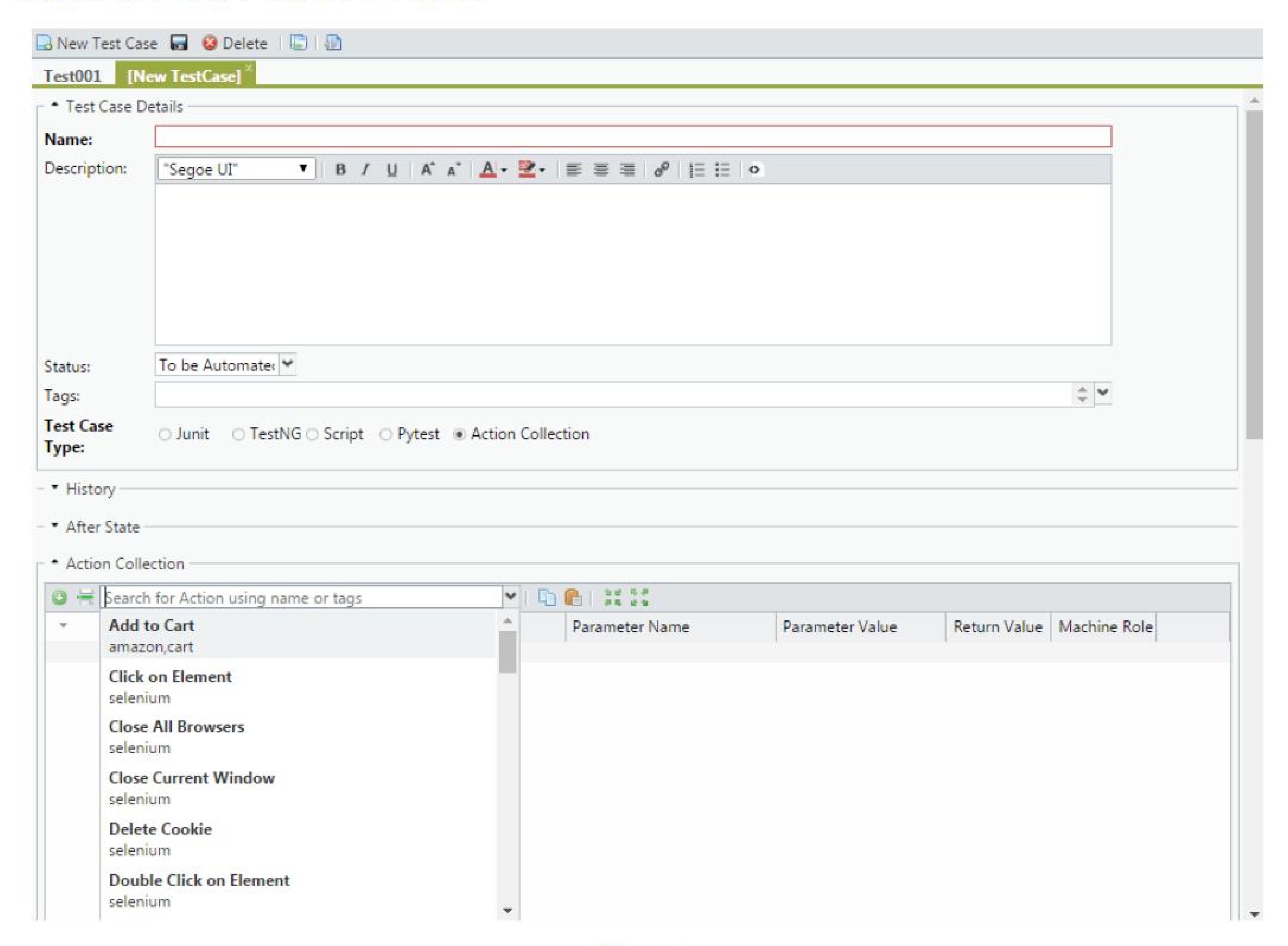


图 8-3-5

切换到 Actions 菜单下，可以看到 RedwoodHQ 已经自带支持的关键字动作，默认已经支持了 Selenium 工具，如图 8-3-6 所示。

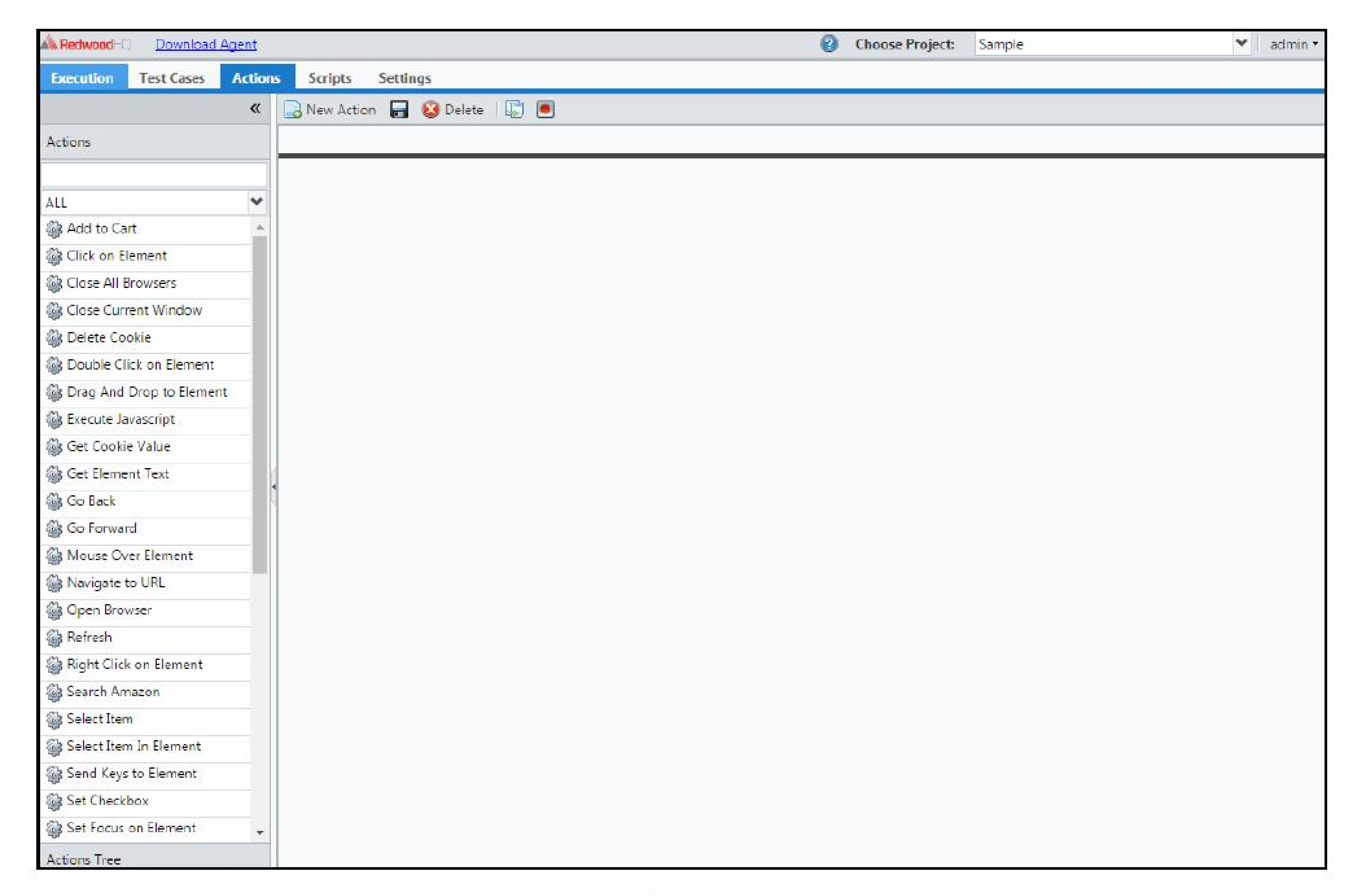


图 8-3-6

通过单击 New Action 按钮，可以新建一个自定义的 Action，如图 8-3-7 所示。在此可以选择 Action 的类型以及可以对自定义编写的 Action 指定标签和对应的实现语言。

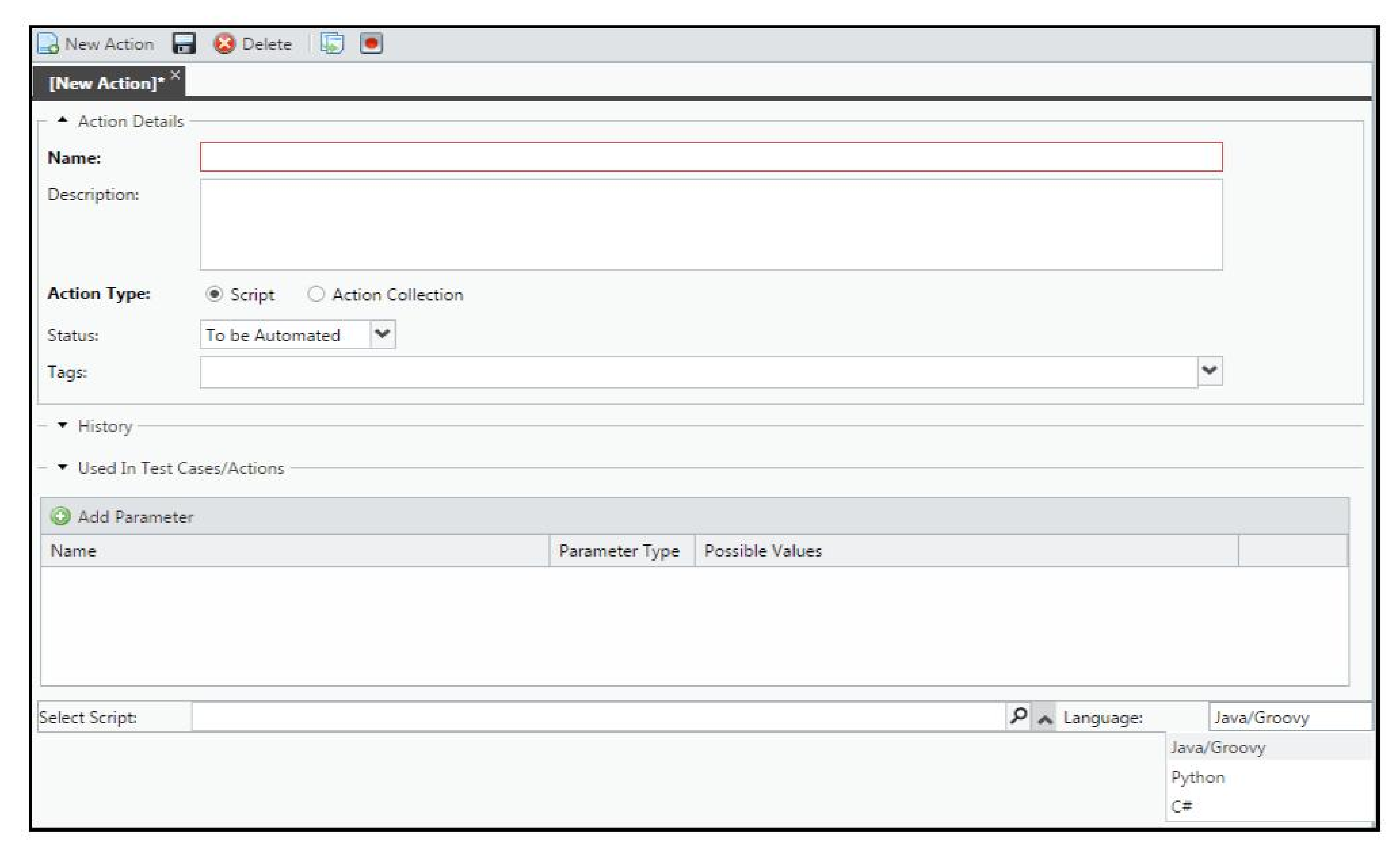


图 8-3-7

表 8-3-1 中列出了两种 Action 类型的详细说明。

表 8-3-1　两种 Action 类型的详细说明

Action 类型	说明
Script	通过测试脚本的方式来实现自定义的 Action，脚本编写支持 Java、Groovy、Python、C#，并且可以指定 Action 执行时需要的参数
Action Collection	选择已有的 Action，与 Robot Framework 中的用户自定义关键字类似，即通过对已有关键字的二次封装构建出一个新的用户关键字

切换到 Scripts 菜单后，可以看到 RedwoodHQ 下的所有脚本，RedwoodHQ 安装好后，已经默认集成了 Selenium 了，并且提供了已经用 Groovy 实现的一部分脚本，如图 8-3-8 和图 8-3-9 所示。

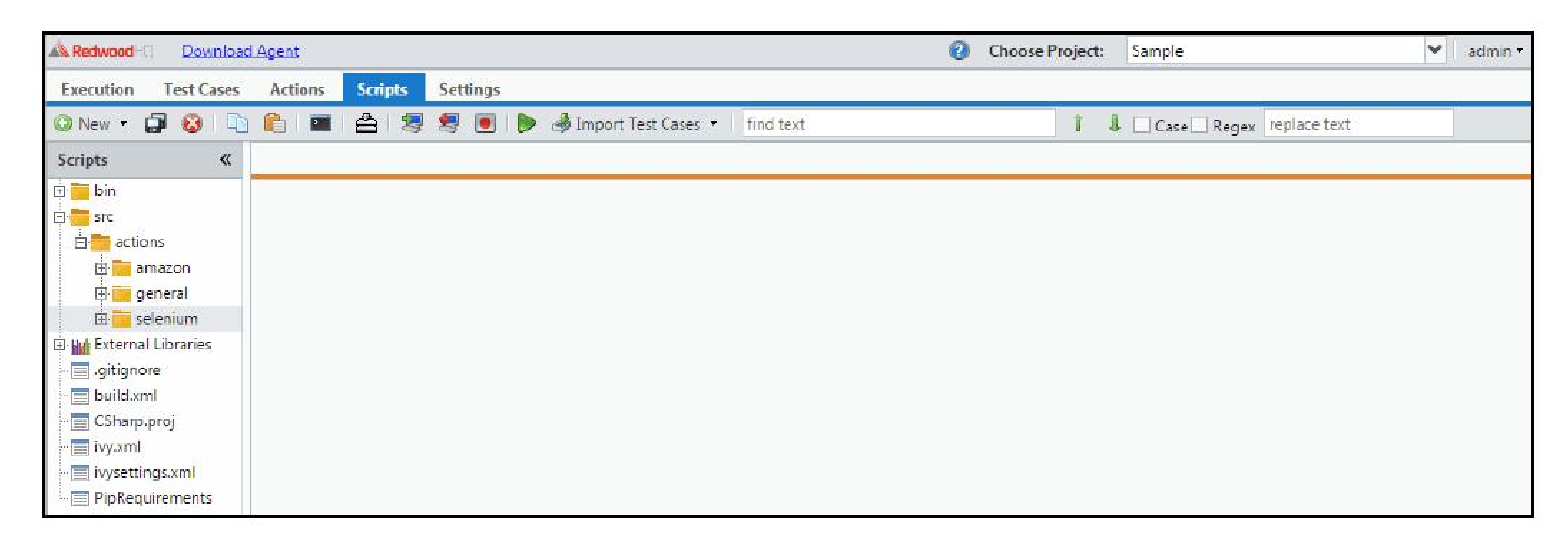


图 8-3-8

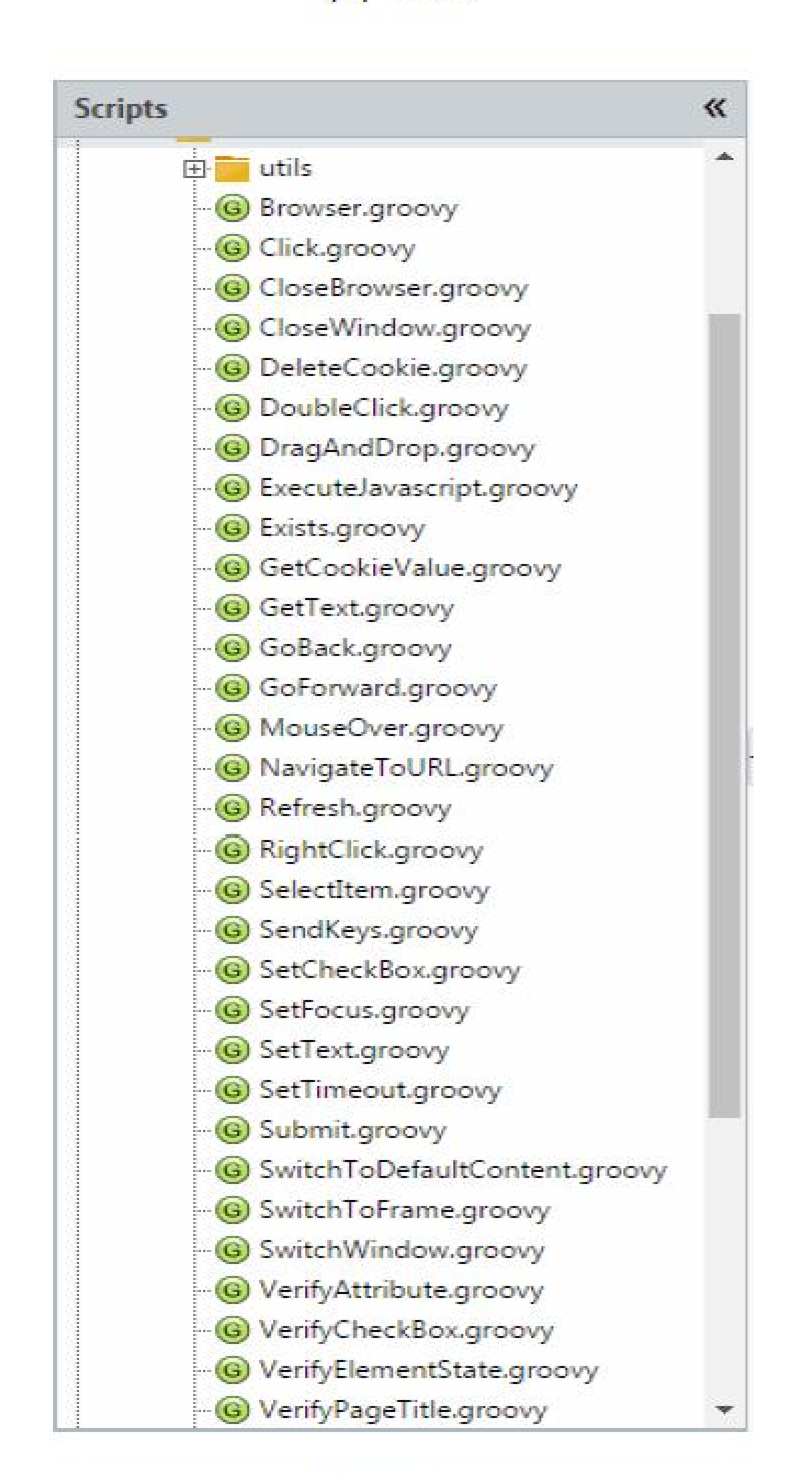


图 8-3-9

选择其中一个脚本文件后，可以查看脚本里面对应的代码实现，如图 8-3-10 所示。可以对已有的脚本进行修改，也可以新建一个自定义的脚本。

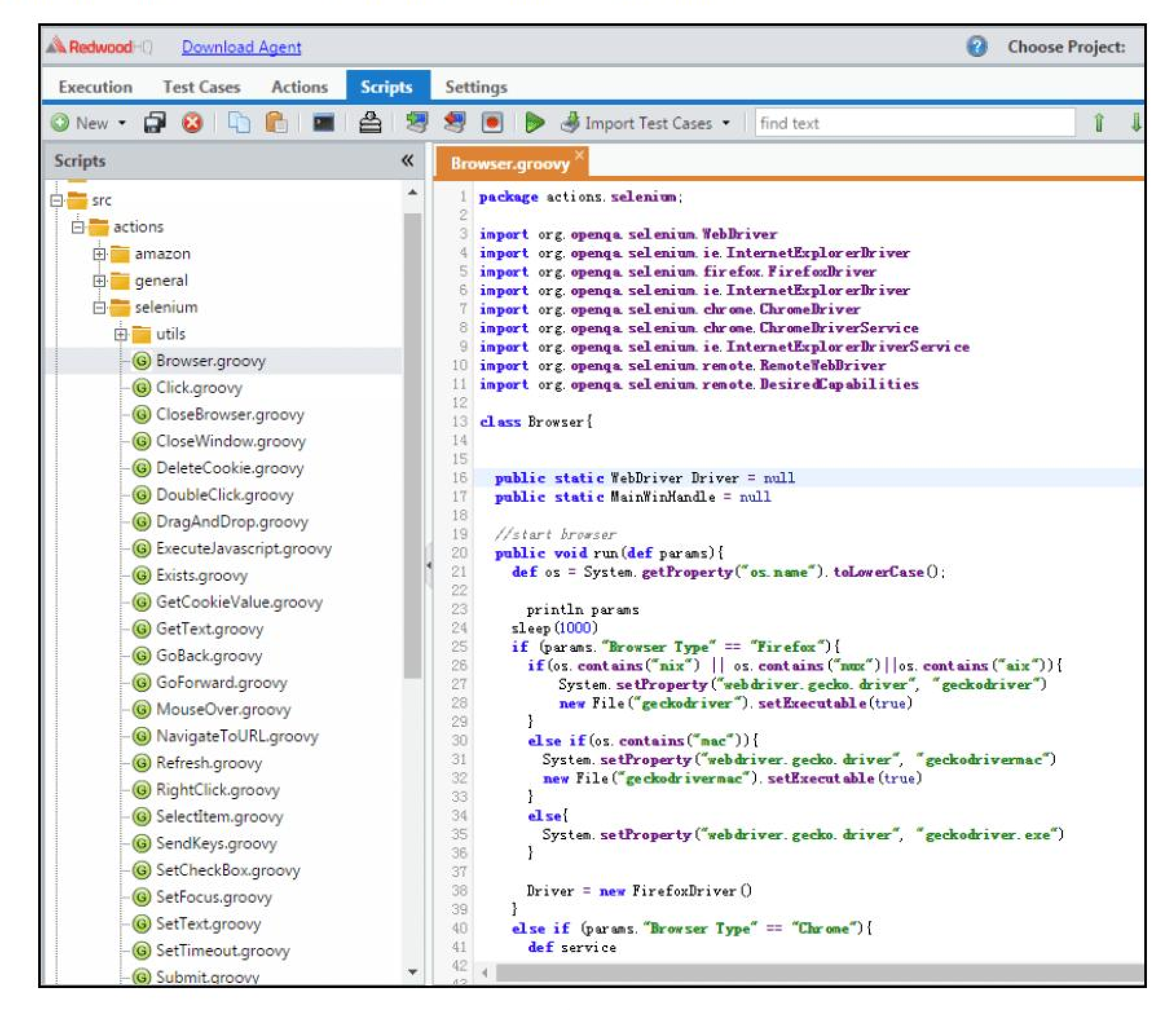


图 8-3-10

新建自定义编写脚本时，可以指定脚本编写实现的语言。这里选择用 Python 语言来实现脚本，如图 8-3-11 所示。

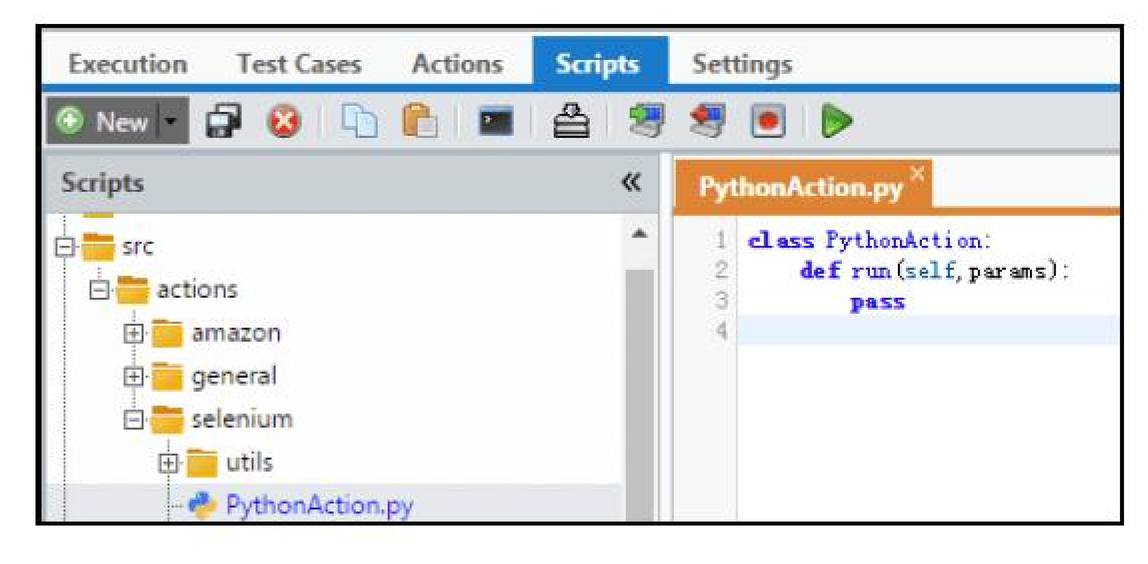


图 8-3-11

8.3.2 Cucumber 介绍

Cucumber 是一款用通俗易懂的普通语言来描述测试用例并且支持行为驱动开发（BDD）思想的开源自动化测试工具。它是一套通过定义 DSL 来验证测试结果的自动化测试框架，支

持 Ruby、Java、.Net、JavaScript 等多种常用的编程语言，由于是基于行为驱动开发的模式，因此这款自动化测试工具很适合敏捷开发模式的团队使用。

通过访问 Cucumber 的官方主页 https://docs.cucumber.io/ 可以看到其相关的官方文档介绍，如图 8-3-12 所示。

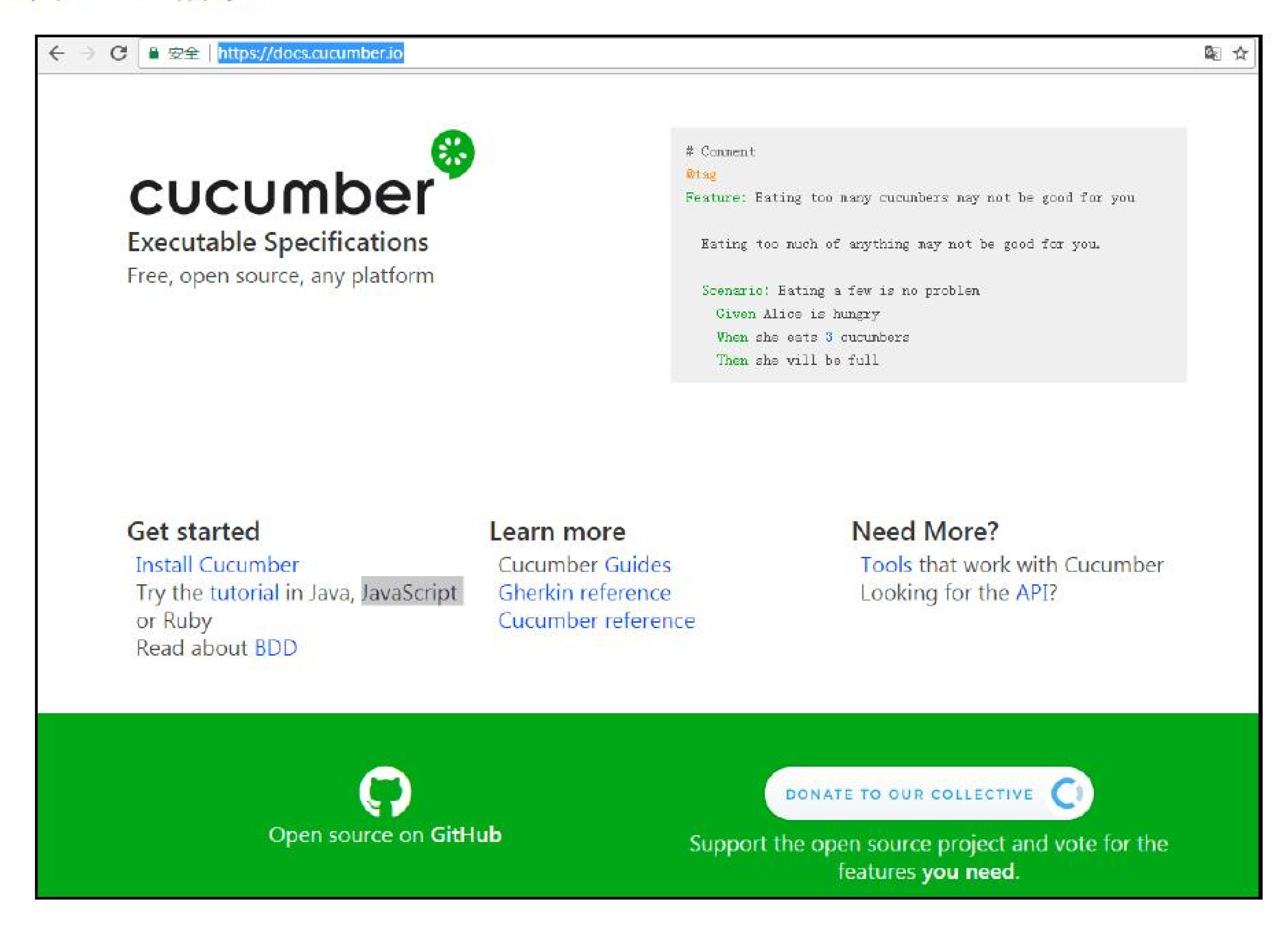
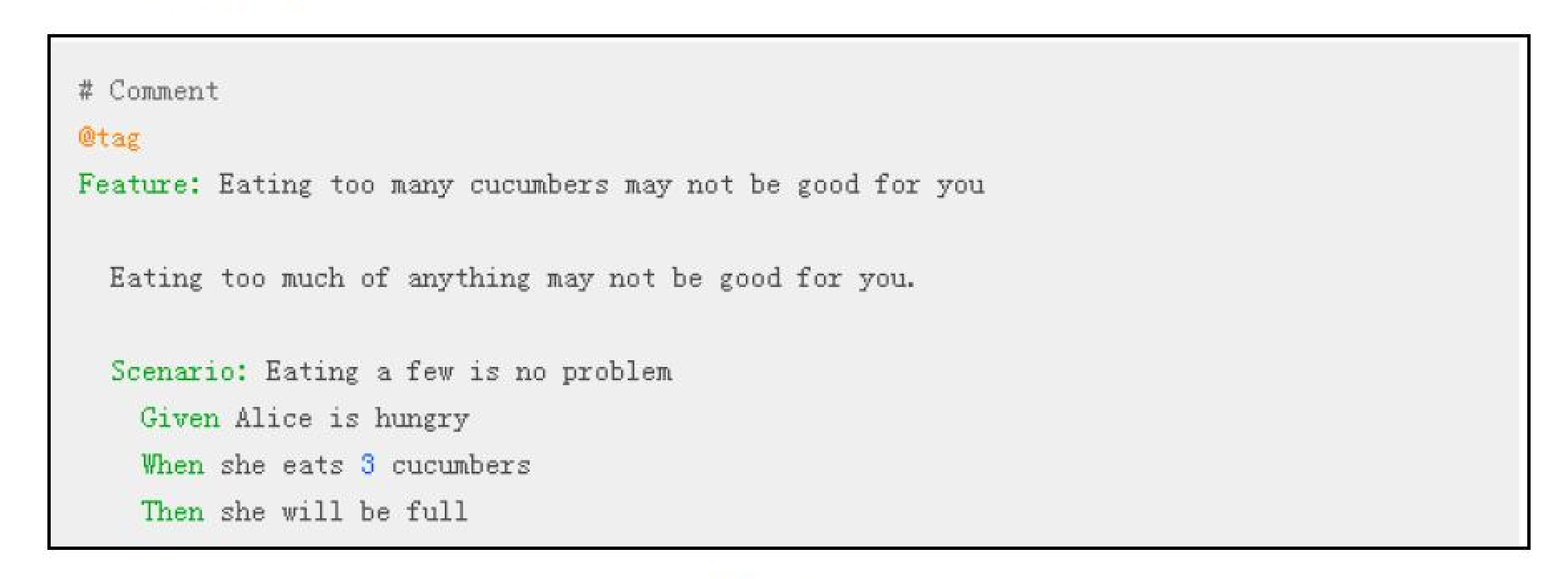


图 8-3-12

在 Cucumber 官方网站上给了一个 DSL 的示例，可以明显看出 Cucumber 行为驱动开发的特点，如图 8-3-13 所示。



```
# Comment
@tag
Feature: Eating too many cucumbers may not be good for you

  Eating too much of anything may not be good for you.

  Scenario: Eating a few is no problem
    Given Alice is hungry
    When she eats 3 cucumbers
    Then she will be full
```

图 8-3-13